

University of Copenhagen

Domain Specific Language for Banking

by

Yuezhu Men &

Andreas Lynge

A thesis submitted in partial fulfillment for the
degree of Master of Computer Science

in the
Faculty of Science
Department of Computer Science

September 2016

Abstract

The Scandinavian Data Center (SDC) company is looking for a new DSL for banks to develop applications accessing SDC's banking services and external services. In order to design such a DSL for SDC, we have analysed user requirements and related languages. We have designed Visual Orchestration Language (VOL), a new DSL for the domain defined by the user requirements. We have evaluated VOL by verifying that it fulfills the user requirements and by comparing it to related languages. VOL fulfills a high majority of the user requirements, and it has a few unique features which are not found in the related languages. To further evaluate VOL, we have implemented BIDE, a Proof of Concept implementation of VOL. By demonstrating BIDE to SDC we got SDC's evaluations of VOL, which are mostly positive. We learned that BIDE is useful for SDC's customers. We were hoping that BIDE would be a useful tool for ordinary bankers, but it turns out that it is more suited for developers at banks and developers at SDC. An interesting result is that we have been able to find a Business Process Model and Notation (BPMN) implementation called Camunda, which might be able to fulfill the same user requirements as VOL. The advantage of using a framework like Camunda is that it already implements BPMN. Hence, to fulfill the user requirements, it is only necessary to implement plugins for Camunda, which is faster than implementing a new DSL. An advantage of using VOL is that VOL is a simpler language than BPMN. VOL can be designed to fulfill the user requirements without limitations of a particular framework. With Camunda it is necessary to accept its limitations.

Contents

List of Figures	v
List of Tables	vii
Abbreviations	viii
1 Introduction	1
1.1 Problem	1
1.2 Procedure	2
1.3 Problem Background	3
1.3.1 SDC's Development Procedure	3
1.3.2 SDC's System	5
1.3.3 The Role of Rules	6
2 Problem Analysis	10
2.1 Problem Decision	10
2.1.1 DSL ideas	10
2.1.1.1 Idea for DSL for replacing Rules	10
2.1.1.2 Idea for DSL for Banks	11
2.1.2 Decision of DSL for Banks	11
2.2 User Requirements	12
2.2.1 Requirements gathering	12
2.2.2 User Stories	13
2.2.3 Requirements for DSL Design	16
2.2.4 Requirements for DSL Implementation	18
2.3 Related Languages	19
2.3.1 Business Process Model and Notation	19
2.3.2 Business Process Execution Language	20
2.3.3 LabVIEW	22
2.3.4 SQL Server Integration Services	23
3 Design	26
3.1 Visual Orchestration Language Design	26
3.1.1 Types	26
3.1.2 Data Flow	28
3.1.3 Control Flow	32
3.2 Evaluation	36

3.2.1	Comparisons with Other Languages	37
3.2.1.1	Comparison with BPMN	37
3.2.1.2	Comparison with Oracle BPEL Process Manager	38
3.2.1.3	Comparison with LabVIEW	39
3.2.1.4	Comparison with SSIS	39
3.2.2	Evaluation with Requirements	40
3.3	Hypothesises	42
3.3.1	VOL is Easy for Bankers	42
3.3.2	ACID Transaction Data Flow	42
4	Implementation	43
4.1	BIDE User Manual	43
4.1.1	Getting Started with BIDE	43
4.1.2	Control Flow	45
4.1.3	Data Flow	51
4.1.4	Data Mapping	53
4.1.5	Example	55
4.1.5.1	Application Implementation	55
4.1.5.2	Application Test	66
4.2	BIDE Internals	70
4.2.1	Abstract Syntax Tree	71
4.2.2	Haskell Representation	71
4.2.2.1	Example of Haskell Representation	71
4.2.2.2	Analysis of Haskell Representation	73
4.2.3	Application Execution	74
4.3	Evaluation	76
4.3.1	Evaluation by SDC	76
4.3.2	Evaluation with Requirements	77
5	Discussion	79
5.1	Comparisons with Related Languages	79
5.1.1	Unique Features of VOL	79
5.1.2	Analysis	80
5.1.2.1	Analysis of SSIS	80
5.1.2.2	Analysis of LabVIEW	80
5.1.2.3	Analysis of BPEL	81
5.1.2.4	Analysis of BPMN	82
5.1.3	Evaluation	82
5.2	Future Work	83
5.2.1	Proof of Concept	83
5.2.1.1	Real Services	84
5.2.1.2	Work on Type System	84
5.2.1.3	ACID Transactions	84
5.2.1.4	Test Hypothesis	85
5.2.2	First Release	85
5.2.2.1	Real Platforms	85
5.2.2.2	Drag and Drop User Interactions	86

5.2.2.3	Implementation Requirements	86
6	Conclusions	87
6.1	Problem	87
6.2	Procedure	87
6.3	Findings	88
A	Grammar Specification	90
B	Grammar Examples	117
B.1	Grammar for US1	117
B.2	Grammar for US4	118
C	Python Tool	119
D	Haskell Server and Platforms	124
E	Java Bank Integrated Development Environment	181
	Bibliography	589

List of Figures

1.1	Overview of SDC's System	2
1.2	SDC's Development Procedure	4
1.3	SDC's System	5
1.4	5-Layer Model	7
1.5	Turbo Cycle Compiler Procedure	8
2.1	SDC's System with BIDE	11
2.2	Business Process Model and Notation Example	20
2.3	Oracle BPEL Process Manager Example	21
2.4	LabVIEW G Example	22
2.5	LabVIEW Controls Panel	23
2.6	SSIS Control Flow Example	24
2.7	SSIS Data Flow Example	25
3.1	VOL Data Flow Units	28
3.2	VOL Data Flow Addition Transformation	29
3.3	VOL Data Flow Fahrenheit to Celsius	29
3.4	VOL Data Flow Read Fahrenheit Write Celsius	30
3.5	VOL Data Flow Accumulate	31
3.6	VOL Data Flow Filter by Direct Debit	32
3.7	VOL Control Flow Read Fahrenheit Write Celsius	34
3.8	VOL Control Flow Apply for Loan	35
4.1	BIDE Start Screen	44
4.2	BIDE File Menu	44
4.3	BIDE Project Explorer	45
4.4	BIDE Multiple Tabs	45
4.5	BIDE Control Flow Units	46
4.6	BIDE Control Flow Interaction Types	47
4.7	BIDE Control Flow Edge Start Marker	47
4.8	BIDE Control Flow Edge Creation	48
4.9	BIDE Control Flow Input Prompt	48
4.10	BIDE Control Flow Select Input	49
4.11	BIDE Control Flow Input Prompt 2	50
4.12	BIDE Control Flow Graph with Start Connected	50
4.13	BIDE Control Flow Select Input 2	51
4.14	BIDE Control Flow All Input Selected	52
4.15	BIDE Data Flow Units	52

4.16	BIDE Data Flow Edge Creation	54
4.17	BIDE Data Flow Data Mapping	54
4.18	BIDE Data Mapping from Record	54
4.19	BIDE Data Mapping from List	55
4.20	BIDE Data Mapping from Two Lists to One List	56
4.21	BIDE Example New Project Prompt	56
4.22	BIDE Example Input Interaction Prompt	56
4.23	BIDE Example Unary Input Dataflow	57
4.24	BIDE Example Control Flow 1	57
4.25	BIDE Example Insert Constant 1	58
4.26	BIDE Example Type Selection	58
4.27	BIDE Example Types	59
4.28	BIDE Example String Constant	59
4.29	BIDE Example Data Flow 1	60
4.30	BIDE Example Insert Output	60
4.31	BIDE Example Data Flow 2	60
4.32	BIDE Example Auto Type Mapping	61
4.33	BIDE Example Data Flow 3 (Finished)	61
4.34	BIDE Example Control Flow Input Prompt	62
4.35	BIDE Example Control Flow Finished Input Prompt	62
4.36	BIDE Example Control Flow 2	63
4.37	BIDE Example Data Flow 4 (Finished)	64
4.38	BIDE Example Control Flow 3 (Finished)	65
4.39	BIDE Example Compilation Problems	66
4.40	BIDE Example Netbank Home Page	67
4.41	BIDE Example Enter Username Page	68
4.42	BIDE Example Task Running Page	68
4.43	BIDE Example Netbank Running Task	68
4.44	BIDE Example Medarbejderportal Waiting Task	69
4.45	BIDE Example Greeting Page	69
4.46	BIDE Example Task Failed Page	70
4.47	BIDE Example Netbank Failed Task	70
4.48	Platform Logon	74
4.49	Start Application	75

List of Tables

2.1	User Requirements for DSL Design	18
2.2	User Requirements for DSL Implementation	19
3.1	VOL Control Flow Units	33
3.2	VOL Control Flow Notions and Corresponding BPMN Components . .	37
3.3	Design Requirement Fulfillment Descriptions	41
4.1	Implementation Requirement Fulfillment Descriptions	78
5.1	Discussion Analysis of VOL	80
5.2	Discussion Analysis of SSIS	80
5.3	Discussion Analysis of LabVIEW	81
5.4	Discussion Analysis of Oracle BPEL Process Manager	81
5.5	Discussion Analysis of Camunda BPMN	82

Abbreviations

ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interface
AST	Abstract Syntax Tree
BIDE	Bank Integrated Development Environment
BPEL	Business Process Execution Language
BPMN	Business Process Model and Notation
CICS	Customer Information Control System
DB	DataBase
DRDA	Distributed Relational Database Architecture
DR	Design Requirement
DSL	Domain Specific Language
ESB	Enterprise Service Bus
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
IDE	Integrated Development Environment
ID	IDentification
IR	Implementation Requirement
JSON	JavaScript Object Notation
MS	MicroSoft
PoC	Proof of Concept
REST	REpresentational State Transfer
SC	Service Composer
SDC	Scandinavian Data Center
SQL	Structured Query Language
SSN	Social Security Number

STS	SDC Transaction Service
TAC	TIA Access Controller
TGW	TIA Generic Web-service
TIA	Transaction Integration Adapter
UML	Unified Modelling Language
URL	Uniform Resource Locator
US	User Story
VOL	Visual Orchestration Language
WSDL	Web Service Description Language
XML	EXtensible Markup Language

Chapter 1

Introduction

The Scandinavian Data Center (SDC) company has specialized in developing and maintaining bank applications and services. SDC offers their customers a banking system with client applications for accessing banking services.

This Master's thesis is based on work we have done in cooperation with SDC, where we have designed a new Domain Specific Language (DSL) for banking.

1.1 Problem

SDC is looking for designs of two new DSLs. On one hand they are looking for a DSL for replacing the Rules language, which is a high level COBOL inspired DSL that lets SDC developers focus on business functionality. On the other hand SDC is looking for a high level DSL that is intuitive for bankers to use for implementing software accessing SDC's services and external services.

Figure 1.1 gives an overview of how SDC's system is designed. As illustrated in the figure SDC's system has an Enterprise Service Bus (ESB) that offers services to banking processes. The Corebank subsystem, which is partly implemented with the Rules language, is used by the ESB for offering services that are accessing the bank database when adding new customers, making money transfers, etc. The ESB also has access to other subsystems offering services.

The Rules language is designed for database communication, but at the same time supporting some general purpose programming constructs. The development and support of the Rules language has ended, and some of SDC's current and potential customers expect SDC to be using technologies with long term support, hence SDC wants a DSL

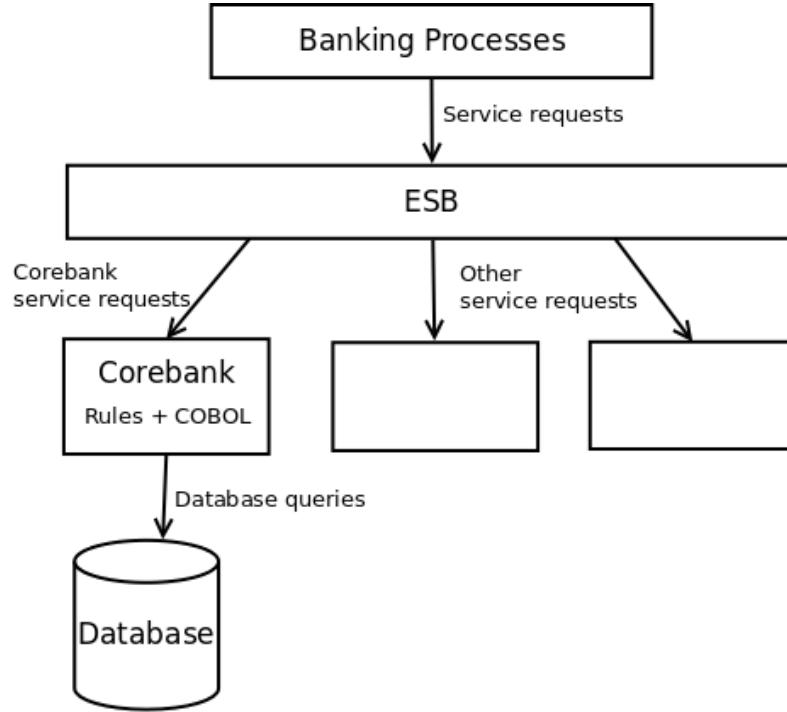


FIGURE 1.1: Overview of SDC's System

with long term support as a replacement for Rules. SDC is not entirely confident in other companies offering long term support, so they consider implementing their own DSL as a replacement for Rules.

SDC does not offer any way that banks can implement programs that use SDC services. Thus, the banking processes in figure 1.1 are implemented entirely by SDC. The problem with this is that the time to market when a bank needs a new program which accesses SDC services is long. If SDC offers a DSL that the banks can use to implement most of the programs accessing SDC services, then the time to market can be decreased for these programs, since design and implementation does not have to go through SDC then.

The objective of this Master's Thesis is to design a new DSL for one of these two domains in cooperation with SDC.

1.2 Procedure

Before we start designing a new DSL, we will decide whether a DSL for replacing Rules or a DSL for banks to access SDC services is the most crucial to design. In order to do so, we need to get an understanding of SDC's internal system and have discussions with SDC.

In the design phase of the new DSL we will critically compare it with other relevant programming languages and discuss pros and cons in cooperation with SDC.

Once we have established an initial DSL design, we will implement a Proof of Concept (PoC) for illustrating important design ideas. We will present the PoC to SDC and potential users in order to get feedback which will be used to evaluate the DSL and improve the design.

1.3 Problem Background

1.3.1 SDC's Development Procedure

In section 1.1 we mentioned that the time to market when a bank wants a new program that accesses SDC services is long. In this section we describe SDC's development procedure when a bank requests a new feature, and we show why the time to market is long.

Figure 1.2 shows the procedure when a bank requests a new feature. In the beginning the bank will contact SDC and request the new feature. Together with the SDC management the bank will discuss and agree on the new feature.

Next, the management will distribute tasks to various departments at SDC. When departments finish subsystems, these subsystems are put in the test environment where they will be tested.

Once all of the subsystems pass the tests, they will be put in the production environment where the bank will have access to the new feature.

To ensure the quality of the work at SDC, many people are involved at each step of the development procedure. This, however, means that each step is slow and the time to market is long.

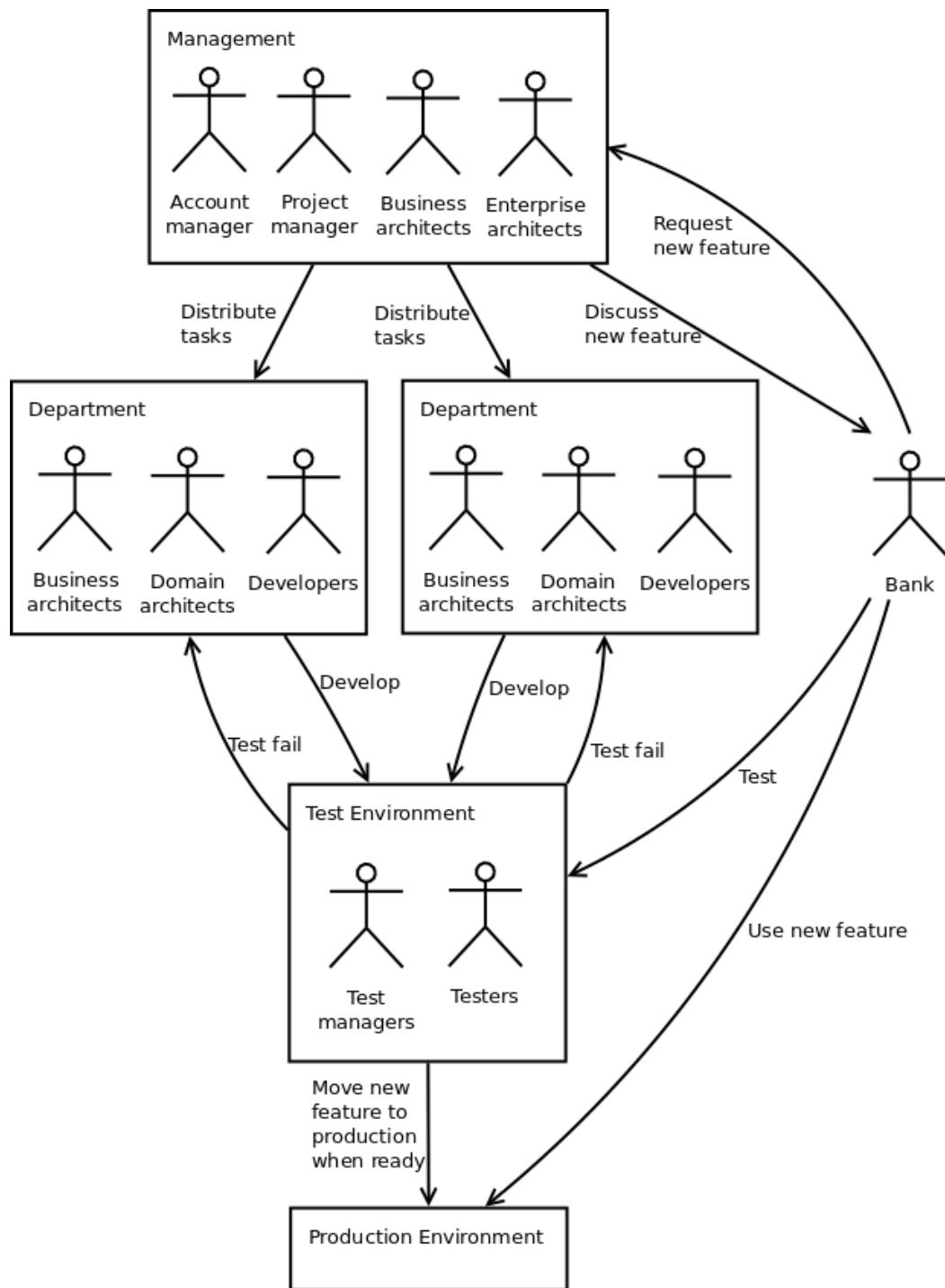


FIGURE 1.2: SDC's Development Procedure

1.3.2 SDC's System

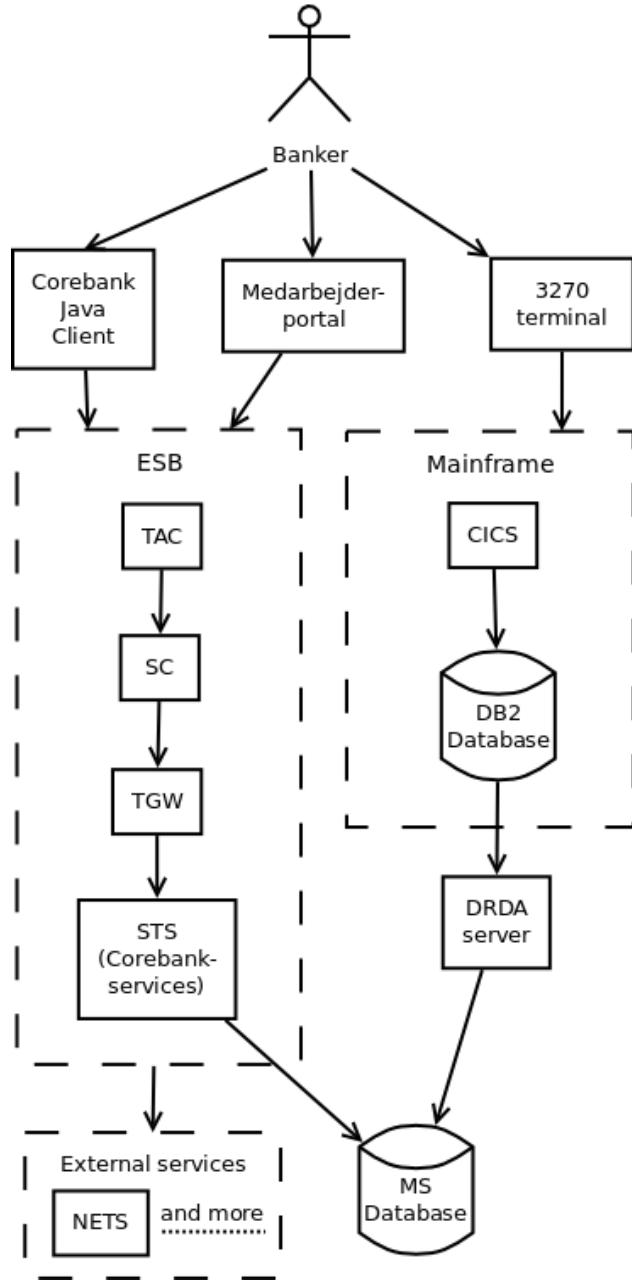


FIGURE 1.3: SDC's System

Section 1.1 gave an overview of SDC's system. Figure 1.3 illustrates SDC's system in more detail.

The figure illustrates that the banks have access to various interfaces accessing SDC's system. The Corebank Java Client and Medarbejderportal is accessing the ESB, and the 3270 terminal emulator is providing a command line interface to the Mainframe. Medarbejderportal is an HTML client, and the Corebank Java Client is partly implemented using the Rules language compiled to Java.

Inside the Mainframe there is a Customer Information Control System (CICS) server that provide ACID transaction management. The 3270 terminal and the CICS server are accessing the DB2 Database for performing queries and updates.

SDC is in a transitional period moving away from using the Mainframe and using the ESB instead. Most data has been moved from being stored on the DB2 Database to now being stored on the MS Database, hence in order to provide the necessary data, the DB2 Database often has to access the Distributed Relational Database Architecture (DRDA) server to indirectly access the MS Database.

Most of the services provided by SDC are now being served by the ESB. Inside the ESB there is a TIA Access Controller (TAC) server, where TIA stands for Transaction Integration Adapter. The TAC server is used for validating client certificates for security. The Service Composer (SC) server is there to offer service orchestrations, by composing multiple services into one service. The TIA Generic Web (TGW) server provides a Web Service Description Language (WSDL) interface to the SDC Transaction Service (STS) server. The STS server is the server which is responsible for providing Corebank services and accessing the MS Database for queries and updates. The ESB is also providing access to a number of external services, it is, for example, providing access to Nets services.

1.3.3 The Role of Rules

To help us decide whether to design a new DSL for replacing Rules or a new DSL for banks, we have to understand the role of the Rules language at SDC. In this section we will explore this.

Rules is a high level data-oriented programming language inspired by COBOL. Rules is used by SDC developers to implement the Corebank Java Client and for implementing STS server Corebank services.

Applications implemented using Rules follow a 5-Layer Model, the 5-Layer Model is illustrated in figure 1.4.

The first 2 layers, the GUI and the Request layers, are Corebank Java Client side layers. The GUI layer implements banking process user interface, and the Request layer implements service requests accessing the STS server.

The last 3 layers, the Transaction; the Operation and the Database layers, are running on the STS server. The Transaction layer implements ACID transaction management,

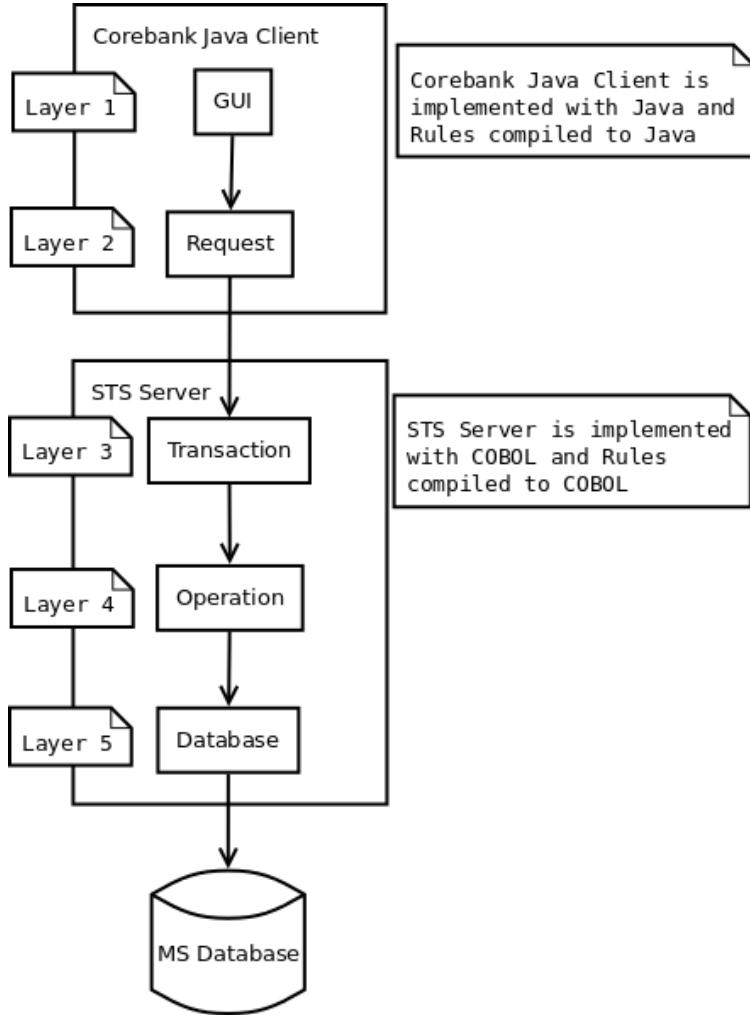


FIGURE 1.4: 5-Layer Model

the Operation layer implements atomic service operations, and the Database layer is responsible for accessing the MS Database.

As we can see in the 5-Layer Model figure, the Corebank Java Client is directly accessing the STS server and is bypassing the TAC, SC and TGW servers. However, the STS server can access the SC services or external services when necessary.

One of the reasons for SDC's success is the Turbo Cycle compiler. The procedure, when implementing Rules programs using the Turbo Cycle compiler, is shown in figure 1.5.

As we can see in the figure, the Turbo Cycle compiler is given a database model and a template as input, given this input it generates Rules code. Next, the SDC developers make additions and changes to the generated Rules code using the App Builder development environment. Once the developers have finished modifying the Rules code, App Builder uses the Rules code to generate DB2 SQL code, COBOL code and Java code. The DB2 SQL is further compiled into MS SQL code such that it can be stored on the

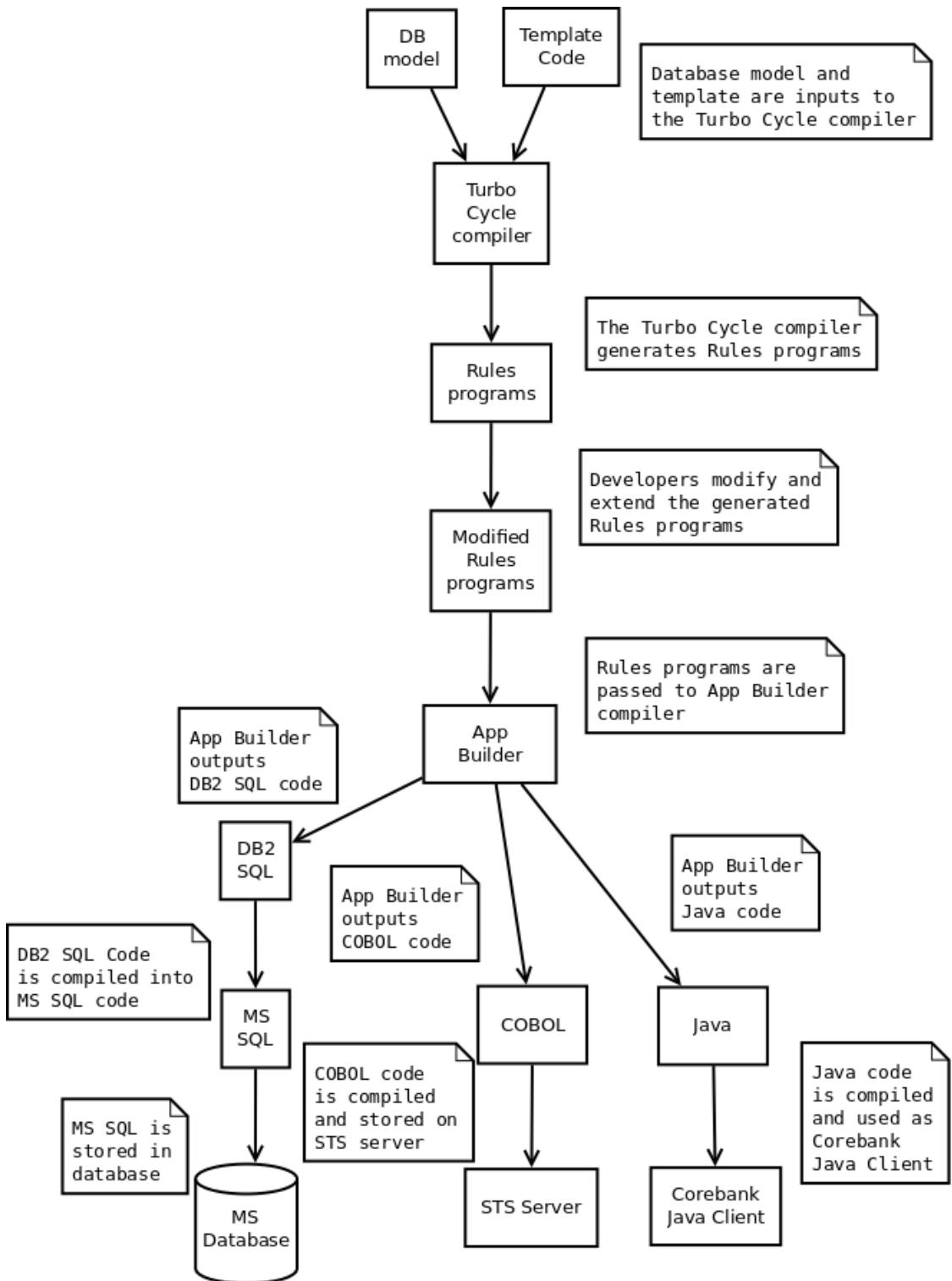


FIGURE 1.5: Turbo Cycle Compiler Procedure

MS Database. The COBOL code generated by App Builder is used by the STS server for implementing the Transaction, Operation and Database layers from the 5-Layer Model. The Java code is used by the Corebank Java Client as layers GUI and Request.

The benefit of the Turbo Cycle compiler is that SDC can generate a lot of Rules code in a short amount of time. The drawback is when the database model or the template given to the Turbo Cycle compiler changes, then new Rules code is generated, and SDC's developers often will have to redo code modifications on the new Rules code, which were made to the previously generated Rules code.

Chapter 2

Problem Analysis

2.1 Problem Decision

After gaining a better understanding of the SDC internals in section 1.3 we understand where the Rules language is used, and we know that the time to market at SDC can be long. In this section we will use this to make two DSL ideas that we can discuss with SDC to determine whether to continue the project by designing a DSL for replacing Rules or a DSL for banks.

2.1.1 DSL ideas

2.1.1.1 Idea for DSL for replacing Rules

For the success of a new DSL for replacing Rules, it is important that the developers at SDC like it, they are going to be the users of the DSL. For that reason, it might be a good idea to design a replacement that is similar to Rules so the language will be familiar to the developers at SDC. If the new DSL is similar and respects the 5-Layer Model from section 1.3.3, then it will also be easier to directly replace Rules code with the new DSL code, if necessary.

If we decide to design a DSL for replacing Rules, we will look into the idea of making it similar, but improve or simplify some features: As described in section 1.3.3, the Turbo Cycle compiler is important to SDC, but if the input to the Turbo Cycle compiler changes, then developers often have to redo code modifications. A new DSL can, for example, improve on this process by not forcing developers to redo code modifications in similar situations.

2.1.1.2 Idea for DSL for Banks

A DSL that banks can use for accessing SDC services can improve on the time to market by letting banks implement programs themselves. This DSL can fit into SDC's system as illustrated in figure 2.1. Alongside the Corebank Java Client, Medarbejderportal and

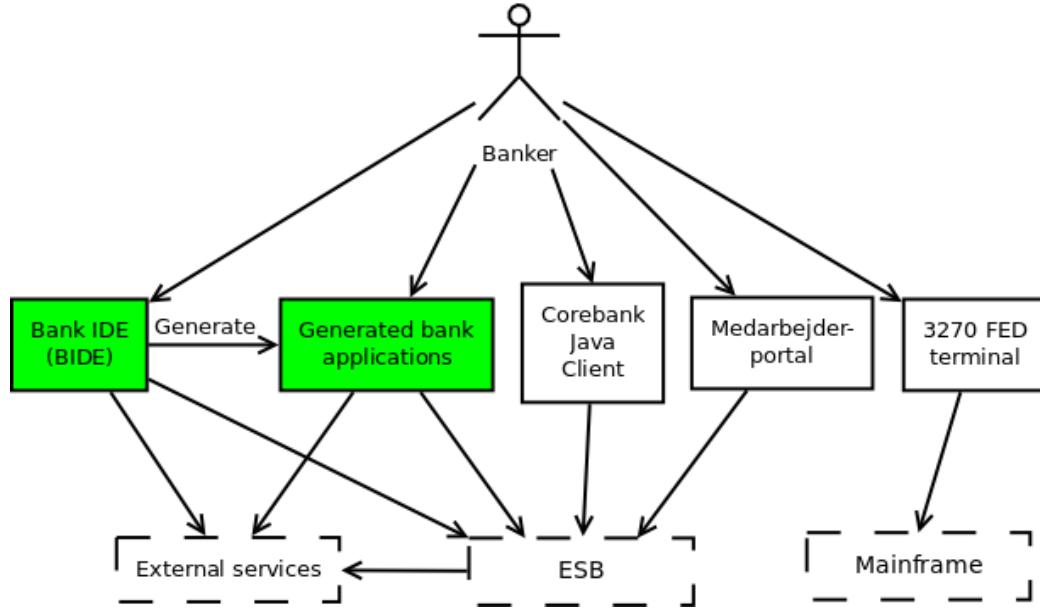


FIGURE 2.1: SDC's System with BIDE

the 3270 terminal, the banks will have a Bank Integrated Development Environment (BIDE) where they can use the DSL and generate bank applications that access the ESB as well as external services.

In order to make the DSL easy to learn for bankers, who are expected to be inexperienced programmers, a text based DSL might not be the best solution. If we decide to design a DSL for banks, we will consider making it a visual (graphical) DSL that is intuitive for bankers who are used to working with Excel Spreadsheets and dragging and dropping files.

2.1.2 Decision of DSL for Banks

We have presented the two ideas to experienced SDC developers and to SDC's technical solution architects manager.

They agree that SDC needs a replacement for Rules, but they disagree on whether a new DSL will be necessary. Some think that it is better to replace Rules with a well known general purpose language like C# rather than implementing a new DSL. Others

think that a good DSL will be worthwhile implementing since it can help developers avoid mistakes.

They all recognize that a DSL for banks to access SDC services is a good idea and that SDC will need something like BIDE in the future. Refer to section 2.1.1.2 for a description of BIDE.

Based on these discussions, since they do not agree on whether a new DSL for replacing Rules is necessary, and they all think that SDC will need something like BIDE in the future, we estimate that a DSL for banks is more crucial to design and has a higher chance of succeeding than a DSL for replacing Rules. Thus, **we have decided to continue by designing a new DSL for banks.**

2.2 User Requirements

This section introduces the user requirements we have gathered for the DSL for banks.

The requirements are kept at a user requirement level rather than a system requirement level, as defined by Ian Sommerville [1, p. 83]. This means that most of the requirements in this section are kept abstract, but when the DSL is designed they can be subdivided into concrete system requirements.

2.2.1 Requirements gathering

Here we will describe how we have gathered user requirements for the DSL. Apart from ourselves, the stakeholders in this project are banks and SDC.

Through meetings with our supervisors at SDC and SDC's technical solution architects manager we have obtained user requirements.

We could have had meetings with banks to gather user stories and user requirements from bankers, but SDC thought that the banks would put pressure on SDC to get the DSL ready if they found out about SDC working on it. Instead we gathered user stories and user requirements by having a meeting with a former banker who is now account manager at SDC. We gathered other user stories by networking with SDC employees and by brainstorming in cooperation with our supervisors at SDC and SDC's technical solution architects manager.

2.2.2 User Stories

This section describes the user stories we have obtained. The user stories are examples of applications that banks would like to implement with the DSL. Each user story has an ID, starting with US, which is used for referencing the user story.

US1 - Develop application which offers a subset of customers with Kapitalpension an Aldersopsparing

As a banker I would like to program an application which:

1. Gets ID of all customers that have a Kapitalpension.
2. Removes ID of customers who do not want to be contacted by the bank.
3. Using the remaining customer IDs, sends email to the corresponding customers offering them an Aldersopsparing.

US2 - Develop button for Netbank where customer can select to receive SMS when account changes

As a banker I would like to program an application which:

1. Has an operation which is performed every time a customer's account changes.
2. The operation should send an SMS to the Netbank customer saying where and when the change happened and how much money were involved.

US3 - Develop application that can calculate a customer's net value

As a banker I would like to program an application which:

1. Gets a customer's dept, bank deposit and property using necessary services.
2. Determines how much of the customer's property the bank owns (x%).
3. Calculates the customers net value using the formula: -dept + bank deposit + $(1 - x\%) \cdot \text{property}$, where x% indicates how much of the property the bank owns.
4. Shows the calculated net value.

US4 - Develop Netbank application which can advice customers

As a banker I would like to program an application which:

1. Gets information about where a given customer spends money.
2. Determines whether the customer can save money somehow, for example by using a different credit card or by using a customer card.
3. Shows suggestions about how the customer can save money.

US5 - Develop customer classification application

As a banker I would like to program an application which:

1. Gets necessary information of a customer.
2. Uses the information for classifying the customer by using some credit score or geographical segmentation.
3. Shows the classification.

US6 - Develop application that gives selected customers a bonus

As a banker I would like to program an application which:

1. Gets all customer IDs of the bank.
2. Gets necessary customer information for each customer ID.
3. Keeps only customers that can be classified as Helkunde. Where Helkunde is a customer who has salary account, a debit card, a savings account, a loan, at least 2 direct debit agreements and is not listed as a poor bill payer.
4. For example, transfers 1000 dkk to each Helkunde or decreases their loans' interest rates.

US7 - Develop an application that can update a reminder letter

As a banker I would like to program an application which:

1. Gets a reminder letter for a customer.
2. Updates the reminder letter by inserting a text section, where the text depends on a classification of the customer.
3. Saves the updated reminder letter.

US8 - Develop an application for adding a new bank customer

As a banker I would like to program an application which:

1. Lets a banker enter a potential customer's SSN.
2. Shows the customer's personal information on the screen, based on the SSN.
3. Allows the banker to modify the personal information on the screen. Notice that the banker only modifies the information on the screen and the information stored in the services is not modified.
4. Lets the banker use the possibly modified information to add the customer as a new customer to the bank.

US9 - Develop application where Netbank user can apply for loan

As a banker I would like to program an application which:

1. Allows a Netbank customer to press a button for applying for a loan.
2. Lets the Netbank customer select a loan amount and a loan duration.
3. Lets a banker receive a message about the Netbank customer applying for the loan.
4. Lets the banker answer by accepting or rejecting the loan.
5. Shows the Netbank customer the banker's answer.

US10 - Develop application which can show statistics about a potential customer

As a banker I would like to program an application which:

1. Lets a banker enter an SSN for a potential customer.
2. Shows statistics about the customer using necessary systems.

The user stories are useful requirements, but they are not very generic. By analysing the user stories, we have obtained further requirements which are more generic than the user stories. The more generic requirements are useful for designing a DSL which can be used for implementing more applications than those described in the user stories. These requirements are described in section [2.2.3](#).

2.2.3 Requirements for DSL Design

In this section we describe the requirements which have impact on the DSL design in chapter 3.

Each row in table 2.1 contains a requirement. The **ID** column is used to assign IDs to requirements, where the ID starts with DR (Design Requirement). The **Sources** column tells where the requirement comes from. The **Description** column is used for describing the requirements.

ID	Sources	Description
DR1	SDC	The DSL should be designed in a way that makes it easy for bankers to use it.
DR2	SDC	The DSL should be a visual (graphical) programming language.
DR3	SDC	The DSL should be designed such that it helps avoiding database and enterprise data inconsistencies.
DR4	SDC	The DSL must support generating applications which can access SDC services and external services.
DR5	SDC	The DSL must support generating multi system workflow applications. For example, an application which starts receiving input from a customer on Netbank and afterwards waits for a banker to answer a question on Medarbejder-portal.
DR6	SDC	The DSL must have support for implementing ACID transactions with service invocations. Long running transactions are not required.
DR7	SDC	The DSL must have error handling.
DR8	SDC	The DSL must support implementation of reusable operations, which can be used by multiple applications implemented with the DSL.
DR9	SDC (account manager)	A TDC scale application is an application which is used when a customer calls the bank. While the banker is talking in the phone with the customer the TDC scale application shows information about the customer. The DSL should allow implementing such applications.

ID	Sources	Description
DR10	SDC (account manager)	<p>A Follow-Up Popup is an application which is like a popup that is shown to some banker. The popup typically asks the banker to answer some question. An example of a Follow-Up Popup message is:</p> <p>”Customer David has turned 50 years old, should we send a bottle of wine to him?”</p> <p>The DSL should support implementing Follow-Up Popup applications, which are started when some event happens.</p>
DR11	US1	DSL developers should be able to use the DSL to implement operations which can send an email.
DR12	US2	DSL developers should be able to use the DSL to implement operations which can send an SMS.
DR13	US1, US6	DSL developers should be able to use the DSL to implement operations which can obtain ID of all of the bank's customers.
DR14	US1	DSL developers should be able to use the DSL to implement operations which can obtain ID of customers who does not want to be contacted.
DR15	US9	DSL developers must be able to use the DSL to add buttons to systems. For example, user should be able to add buttons to Netbank pages.
DR16	US9	DSL developers must be able to use the DSL to implement operations which are executed when a particular button is clicked.
DR17	US6	DSL developers should be able to use the DSL to implement operations which can transfer money.
DR18	US7	DSL developers should be able to use the DSL to implement operations which gets a reminder letter for a customer.
DR19	US7	DSL developers should be able to use the DSL to implement operations which saves a reminder letter for a customer.
DR20	US8, US10	DSL developers should be able to use the DSL to implement operations which use a SSN for obtaining information about the person.
DR21	US8	DSL developers should be able to use the DSL to implement operations which add a new bank customer.

ID	Sources	Description
DR22	US8	DSL developers should be able to use the DSL to implement programs which can show data on the screen and afterwards allow the data on the screen to be modified and used.
DR23	US3, US4, US5, US6, US8, US10	DSL developers must be able to use the DSL to implement operations which can obtain information about a customer using SDC services or external services.
DR24	US3	DSL developers must be able to use the DSL to implement math operations.
DR25	US1, US6	DSL developers must be able to use the DSL to implement data transformation operations, transforming one data set into another.
DR26	US2	DSL developers should be able to use the DSL to implement operations which are executed when an event happens. For example, when a customer's account changes or when a customer turns 50 years old.

TABLE 2.1: User Requirements for DSL Design

2.2.4 Requirements for DSL Implementation

Here we list the requirements which are concerned with the DSL implementation rather than the DSL design. The ID of these requirements starts with IR (Implementation Requirement). See table 2.2.

ID	Sources	Description
IR1	SDC	The DSL must support generating applications that are started from different platforms, for example Medarbejder-portal or Netbank.
IR2	SDC	The DSL must support generating applications which can be used by all users of a platform.
IR3	SDC	The DSL must support generating applications which can only be started by selected users of a platform.
IR4	SDC	The DSL implementation must support restricting access to selected employees, so only selected employees can use the DSL.
IR5	SDC	The DSL implementation must support SDC adding reusable operations to all banks or selected banks.

ID	Sources	Description
IR6	SDC	A subset of banks must be able to share an application implementation, which is implemented with the DSL.
IR7	SDC	The DSL implementation must support a deployment mechanism where applications are put in a test environment before production environment.
IR8	SDC	The DSL implementation must support SDC implementing an application which can be used by all banks or selected banks.
IR9	SDC	The DSL implementation must support SDC adding services which are available to all banks or selected banks.
IR10	SDC	The DSL implementation should support banks adding external services which are only available to that bank.
IR11	SDC	The DSL Implementation must be capable of accessing documentation of available services.
IR12	SDC	The DSL implementation and generated applications must access SDC services and external services in a secure way, confirming to security policies.

TABLE 2.2: User Requirements for DSL Implementation

2.3 Related Languages

Now that we have obtained user requirements for the DSL, we will research related languages which we will use for inspiration and experience. The user requirements suggest that we need to research business process modelling languages as well as visual programming languages.

2.3.1 Business Process Model and Notation

Business Process Model and Notation (BPMN) is a standard for business process modeling that provides a graphical notation for specifying business processes. The objective of BPMN is to support business process management, for both technical users and business users, by providing a notation that is intuitive to business users, but still capable of representing complex process semantics. [2]

We describe BPMN here because we know, from a meeting at SDC, that it is common that banks have experience with BPMN.

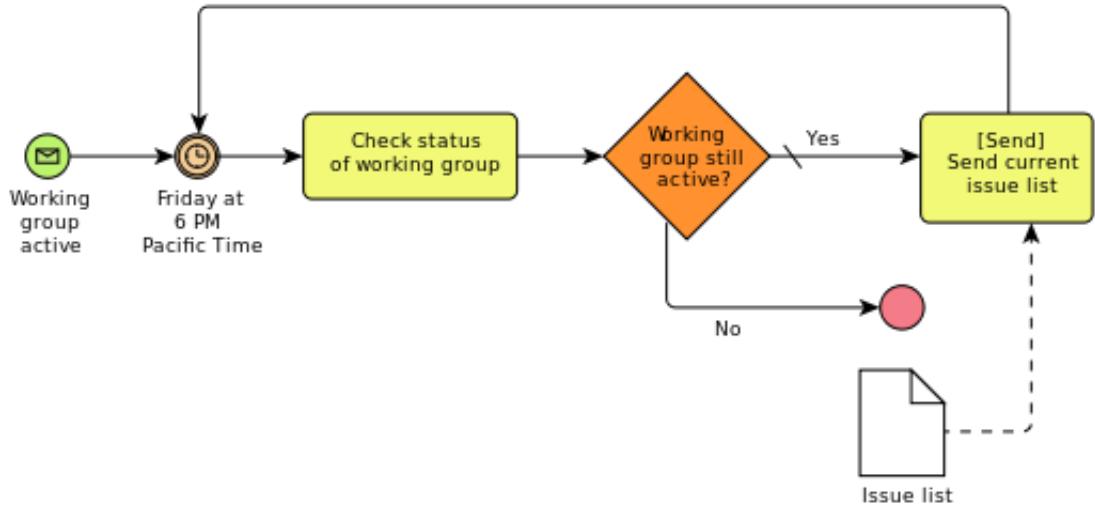


FIGURE 2.2: [2] Business Process Model and Notation Example

Figure 2.2 shows an example of a BPMN process diagram. As we can see in the figure, the process diagram has similarities with the UML activity diagram [3].

The example in the figure models a business process for sending an issue list to a working group. It models that every Friday the business process gets the status of the working group. If the working group is still active the process sends an issue list and waits for next Friday to repeat itself. Otherwise, if the working group is not active anymore the process will end.

By drawing on inspiration and experience from BPMN we hope to design a language that bankers will feel familiar with.

In section 3.2.1.1 we compare our DSL design with BPMN.

2.3.2 Business Process Execution Language

We heard about Business Process Execution Language (BPEL) at a meeting at SDC. It is an XML based executable language for specifying actions within business processes using web services [4]. BPEL is an executable orchestration DSL similar to the kind of DSL we are designing. BPEL processes import and export information by using web service interfaces, such as WSDL interfaces.

The Oracle BPEL Process Manager offers a GUI for graphically creating, deploying and managing BPEL business processes [6]. Figure 2.3 provides an example of a business process modelled with the Oracle BPEL Process Manager.

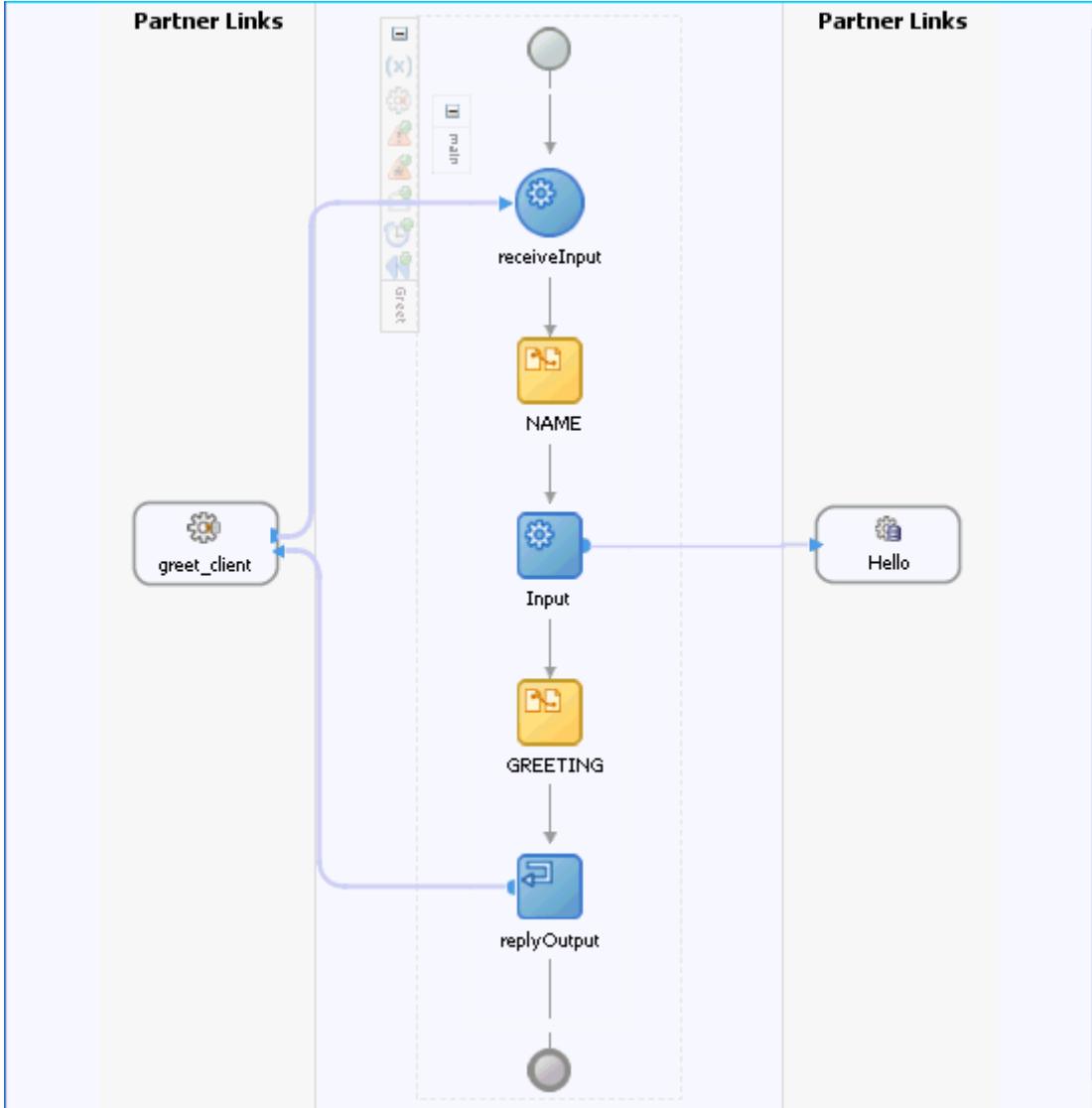


FIGURE 2.3: [5] Oracle BPEL Process Manager Example

The figure illustrates how to implement a simple service using a business process. The Partner Link to the left (`greet_client`) is a service requester, it passes the service input to the business process. The business process uses the input for passing it to an assign activity called `NAME`, an assign activity is essentially a data flow. Once the `NAME` assign activity has finished, the service invocation called `Input` invokes a service provider called `Hello` by passing the output from the `NAME` assign activity as argument. The result from the service invocation is in turn passed to another assign activity called `GREETING`. In the end, the result of the `GREETING` assign activity is used as service response.

We research Oracle BPEL Process Manager in order to learn about designing visual orchestration languages.

Section 3.2.1.2 contains a comparison between our DSL design and Oracle BPEL Process Manager.

2.3.3 LabVIEW

LabVIEW is an IDE designed for building measurement and control systems using a visual programming language called G [7]. We discovered LabVIEW through online research. It is interesting to us because G is a visual DSL and because it has been an actively developed commercial product for a long time, since 1986 [8].

Figure 2.4 demonstrates the G programming language with an example which can convert Fahrenheit degree values to Celsius degree values. The **Temperature (F)** component is representing a slider which provide Fahrenheit degree input values for the program. A Fahrenheit value is read from the **Temperature (F)** component and used for subtracting 32 from it. Then the result of the subtraction is multiplied with $\frac{5}{9}$, the result of the multiplication is a Celsius value converted from the Fahrenheit value. The Celsius value is used as input to the **Thermometer (C)** component for visualizing the Celsius value in a thermometer.

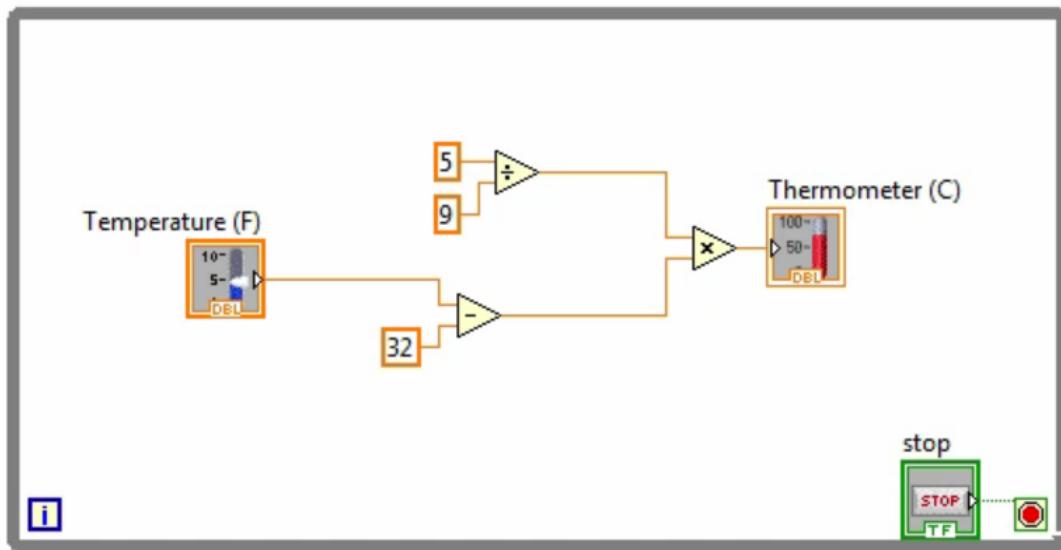


FIGURE 2.4: [9] LabVIEW G Example

The **stop** component in the bottom right of the figure is representing a **stop** button. The gray box around all the components is representing a loop which only stops once the **stop** component provides an output, when the stop button is pressed. So the program continuously reads a Fahrenheit value from the **Temperature (F)** component, converts it to a Celsius value and displays the Celsius value in the **Thermometer (C)**

thermometer. When the **stop** button is pressed, the **stop** component provides an output and the program will terminate.

Figure 2.5 shows the GUI containing the **Temperature (F)** slider, the **Thermometer (C)** thermometer and the **stop** button which is implemented with drag and drop. In the figure we can see that 101.504 Fahrenheit degrees is equivalent to 38.6132 Celsius degrees. The program allows the user to change the **Temperature (F)** slider value which will result in the **Thermometer (C)** value changing correspondingly.

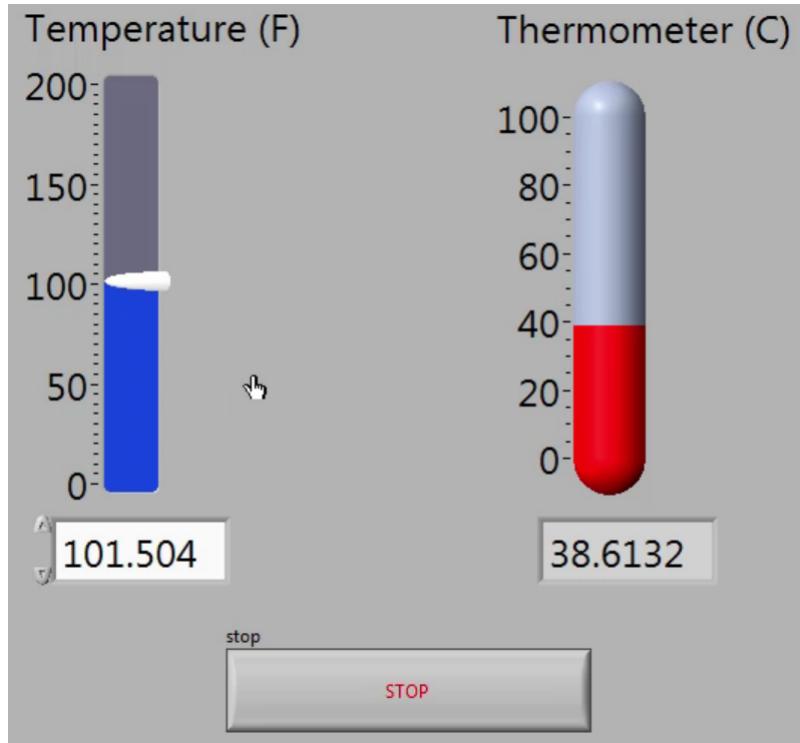


FIGURE 2.5: [9] LabVIEW Controls Panel

LabVIEW is an important source of inspiration to design a visual data flow DSL. In section 3.2.1.3 we compare our DSL design with LabVIEW.

2.3.4 SQL Server Integration Services

We were introduced to SQL Server Integration Services (SSIS) by a supervisor at SDC. SSIS is a visual tool from Microsoft for building data integration and data transformation applications [10]. SSIS is often used for a read-transform-write pattern which extracts and transforms data from sources, such as databases and Excel files, and then stores the processed data on some target system, such as a data warehouse. Except from user stories US2 and US9, a read-transform-write pattern is what all the user stories from section 2.2.2 request.

SSIS applications mainly consist of a control flow and a set of data flow tasks. The control flow is the entry point of the SSIS application. The control flow contains data flow tasks or other tasks which can, for example, send emails or execute SQL statements.

Figure 2.6 shows a simple SSIS control flow. It starts by executing an SQL statement which removes all the rows from the database table `tblSeries`. Then it executes a data flow which reads all the British X-Factor winners from Excel files and stores them in the `tblSeries` table in the database.

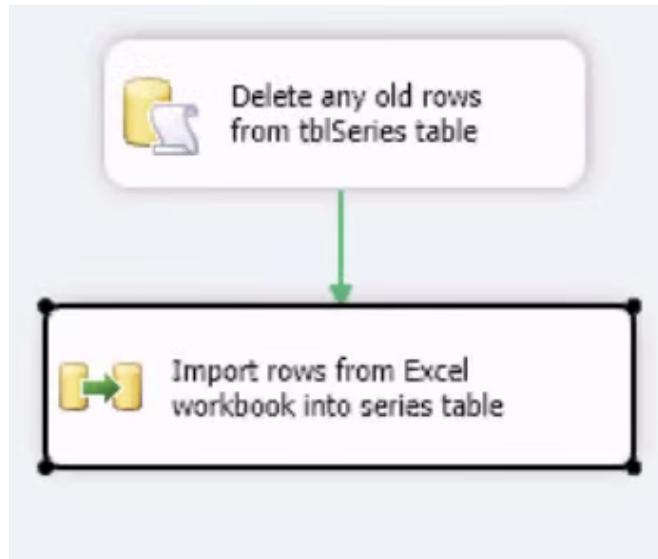


FIGURE 2.6: [11] SSIS Control Flow Example

The data flow is shown in figure 2.7.

The data flow reads all the male X-Factor winners from one Excel file and reads all the female X-Factor winners from another Excel file. Then it passes the male and female winners to a union transformation which outputs a list of all the X-Factor winners. In the end, all the X-Factor winners are stored in database table `tblSeries`.

SSIS is of interest to us because it is a widely used visual tool with user stories corresponding to the user stories we have gathered in section 2.2.2.

In section 3.2.1.4 we compare our DSL design with SSIS.

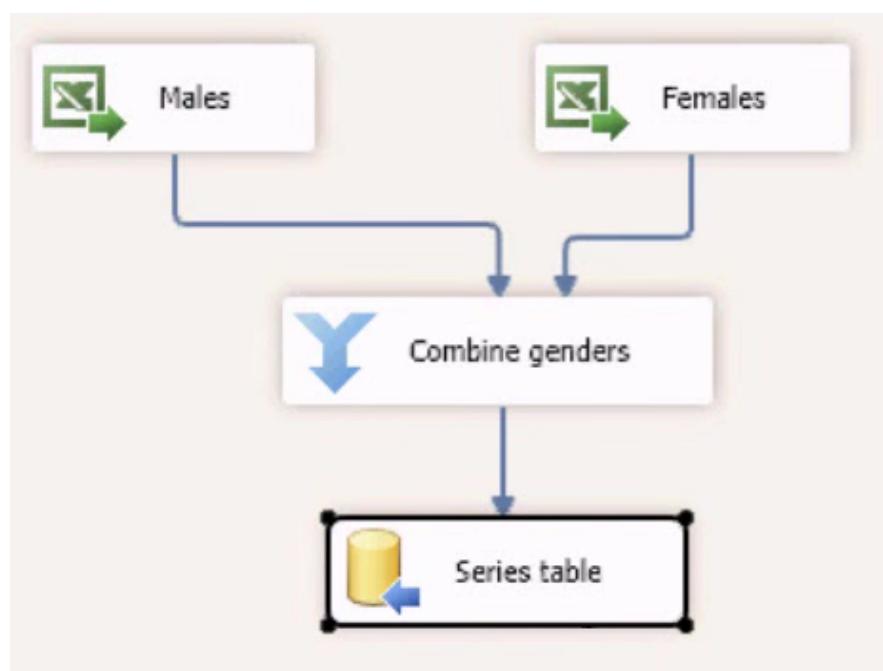


FIGURE 2.7: [11] SSIS Data Flow Example

Chapter 3

Design

This chapter describes and analyses the design of the DSL for banks to access SDC services and external services. The DSL, we have designed in cooperation with SDC, is called Visual Orchestration Language (VOL).

3.1 Visual Orchestration Language Design

In this section we look at the design of VOL. We have attached a complete grammar specification for VOL in appendix A. We have attached comprehensive grammar examples in appendix B which strictly follow the grammar specification.

The design of VOL is used as requirements for our implementation of VOL, called BIDE, in chapter 4.

VOL is an orchestration language where applications implemented with VOL coordinate interconnections and interactions between systems and services. VOL is characterized by clearly distinguishing between control flow and data flow parts of the language, similar to SSIS described in section 2.3.4.

The design of VOL is constructed through a process of iterations with evaluations. It is evaluated with design requirements and comparisons with other languages.

3.1.1 Types

VOL is a statically typed language. It knows about 6 nullary types and 2 type constructors. This section describes this using text syntax, but a language implementation may choose to use a graphical representation of types.

The 6 nullary types supported by VOL are:

1. **Number**
2. **Boolean**
3. **String**
4. **Date**
5. **Time**
6. **Error**

Examples of **Number** constants are 0, 3.14 and -42.

Boolean values can have one of two values: **True** or **False**.

String values are strings of UTF-8 text, for example, "Hello", "Hello, World" and "0".

Date constants have the format YYYY-MM-DD, for example, 2016-09-23.

Examples of **Time** constants are 0 day and 100 millisecond.

The values of **Error** types have an error code, and an error message. The identifier before the pair of parenthesis represents the error code. Inside the parenthesis is an error message of type **String**. Examples of **Error** constants are:

`Conversion_error("Cannot cast abc to Number")` and `Invalid_input("0 < 1")`.

The 2 type constructors in VOL are:

1. **List**
2. **Record**

The **List** type constructor takes a single type as argument. We can, for example, represent a list of numbers type as follows: `List(Number)`, and we can represent a list of lists of strings as follows: `List(List(String))`.

An example of a `List(Number)` constant with 3 elements is: [0, 1.5, 42]. An example of a `List(String)` constant with 2 elements is: ["Hello,", "World"].

The **Record** type constructor takes 1 or more named type arguments. We can represent a point type as follows: `Record(x:Number, y:Number)`. Another example of a type constructed from a **Record** is: `Record(name:String, ssn:Number, accounts:List(Number))`.

An example of a `Record(name:String, accounts>List(Number))` constant is:
`{name:"Tony Hoare", accounts:[01234, 56789]}`.

Types are used for input ports and output ports. As we will see in section 3.1.2, output ports are connected to input ports, but connecting an output port to an input port is only allowed if the ports have the same type. All ports must have a type.

3.1.2 Data Flow

The data flow parts of VOL consist of data flow units which are connected with directed edges. Data flows visualize data dependencies between data flow units. The possible data flow units are shown in figure 3.1.

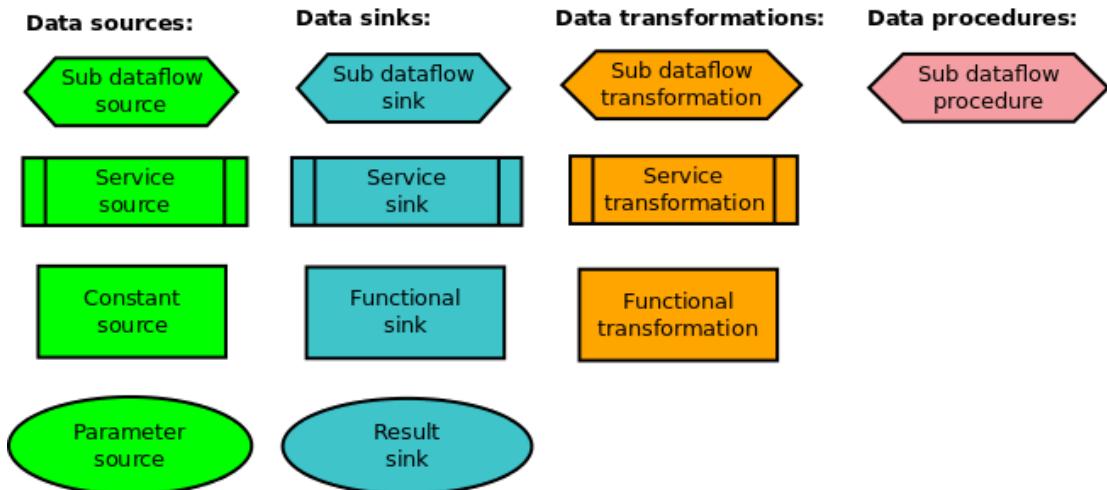


FIGURE 3.1: VOL Data Flow Units

There are 4 groups of data flow units: source units, sink units, transformation units and procedure units. Source units are defined by having only output ports, sink units are defined by having only input ports, transformation units are defined by having both input and output ports, and procedure units are defined by not having any ports.

All ports must be identified by a name. For example, the transformation in figure 3.2, for performing mathematical addition, has 2 input ports and a single output port. One input port is identified by `Left`, the other input port is identified by `Right`, and the output port is identified by `Result`. Note that the port types are hidden in the figure, but all the ports in the figure have type `Number`.

The `Service source`, `Service sink` and `Service transformation` units are used for service invocations. For example, for invoking SDC services or external services.

The `Constant source` unit is used for defining data constants.

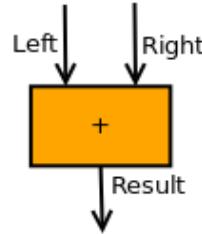


FIGURE 3.2: VOL Data Flow Addition Transformation

The **Functional sink** unit is used for loops which do not have a result.

The **Functional transformation** unit is used for data processing.

The **Sub dataflow source**, **Sub dataflow sink**, **Sub dataflow transformation** and **Sub dataflow procedure** units are used for data flow invocations, i.e. while inside one data flow these units can be used to execute another data flow.

The **Parameter source** unit is used for specifying data flow inputs.

The **Result sink** unit is used for specifying data flow outputs.

Data flows typically follow a read-transform-write pattern. They read data from sources, apply transformations to the data, and then write the transformed data to sinks.

A data flow example, inspired by LabVIEW figure 2.4, which can convert a Fahrenheit degree to a Celsius degree, is given in figure 3.3.

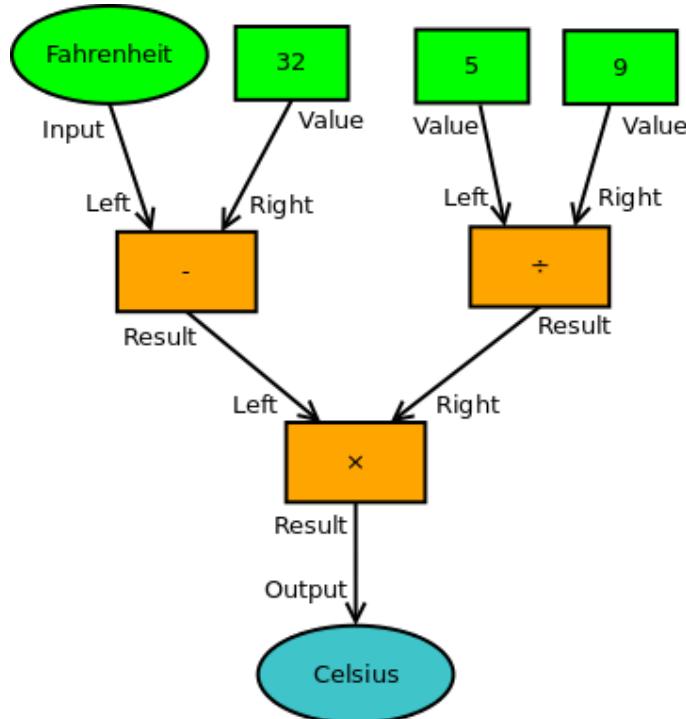


FIGURE 3.3: VOL Data Flow Fahrenheit to Celsius

Provided that the data flow for converting a Fahrenheit degree to a Celsius degree is called `Fahrenheit_to_Celsius`, we can implement a data flow which reads a Fahrenheit degree from a service, converts the Fahrenheit degree to a Celsius degree and writes the Celsius degree to another service, as illustrated in figure 3.4.

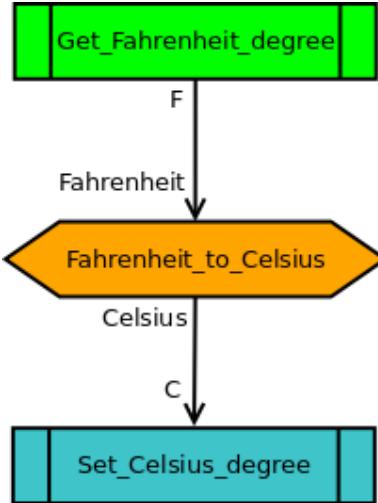


FIGURE 3.4: VOL Data Flow Read Fahrenheit Write Celsius

How a data flow gets associated with an identifier like `Fahrenheit_to_Celsius` is implementation defined.

When processing lists of data it is often necessary to operate on the list elements separately. For this reason, some `Functional transformation` and `Functional sink` units must be associated with a data flow.

Figure 3.5 shows an example of an `Accumulate` transformation, which is used to implement a data flow, which can compute the sum of a list of numbers.

The figure illustrates that the `Accumulate` transformation is associated with the data flow inside the dotted rectangle. A VOL implementation may represent this association in another way, for example, by double clicking the `Accumulate` transformation the user can be taken to a window containing the associated data flow.

The `Accumulate` transformation takes a list (`List`) and an initial accumulator value (`Accum`) as input, the types of `List` and `Accum` are defined by the associated data flow input types. The associated data flow is invoked once for each element in the list. The associated data flow also has two inputs, an element from the list (`Item`) of type `Number` and the accumulated value from the previous iteration (`Accum`) of type `Number`, where the accumulated value in the first iteration is the accumulator value which was given as `Accum` input to the `Accumulate` transformation. The result of the `Accumulate`

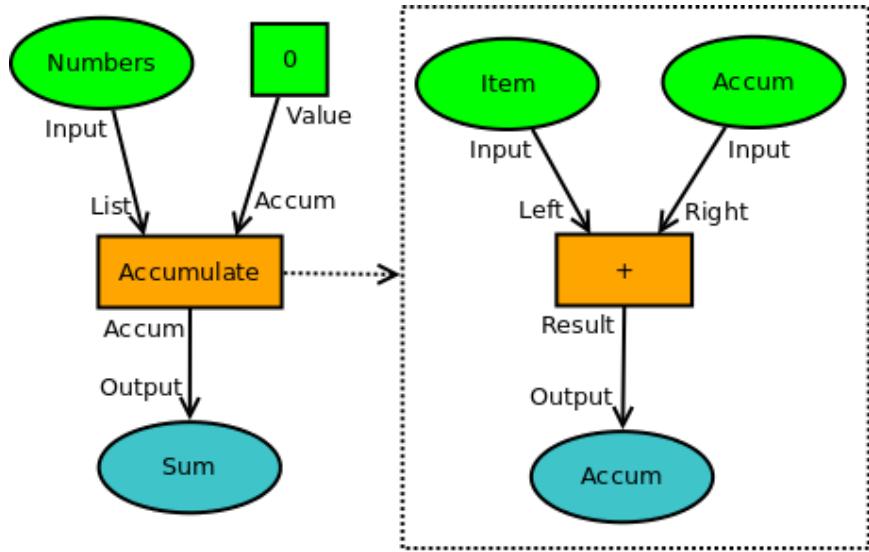


FIGURE 3.5: VOL Data Flow Accumulate

transformation is the associated data flow's accumulated output value (**Accum**) from the last iteration.

Figure 3.6 shows another example of a transformation which must be associated with a data flow. Provided a banking service called `Get_Direct_Debit_Agreements` for obtaining a list of a customer's direct debit agreements, the data flow to the left in figure 3.6 shows how to implement a data flow for filtering a list of customers by keeping the customers who have at least 1 direct debit agreement.

The data flow to the left in the figure has one input called `Customers` of type `List(Number)`. The data flow has one output called `DD_customers` also of type `List(Number)`. Since a customer is represented by a `Number`, we can think of the data flow as having a list of customers as input and a list of customers as output.

The **Filter** transformation is associated with the data flow inside the dotted rectangle. The associated data flow inside the dotted rectangle has one input called `Item` of type `Number` and one result called `Keep` of type `Boolean`. The associated data flow uses the `Item` input, which in this case is a customer represented by a `Number`, for obtaining a list of direct debit agreements. Next, the length of the list of direct debit agreements is compared to 0. If the length is greater than 0, the associated data flow outputs `True`, otherwise it outputs `False`.

The **Filter** transformation works as one would expect: For each item in the **Filter**'s input list, the associated data flow is invoked by passing that element as input. If the associated data flow returns `True`, then the element is added to the **Filter**'s output list (`Result`), otherwise the element is not added to the **Filter**'s output list.

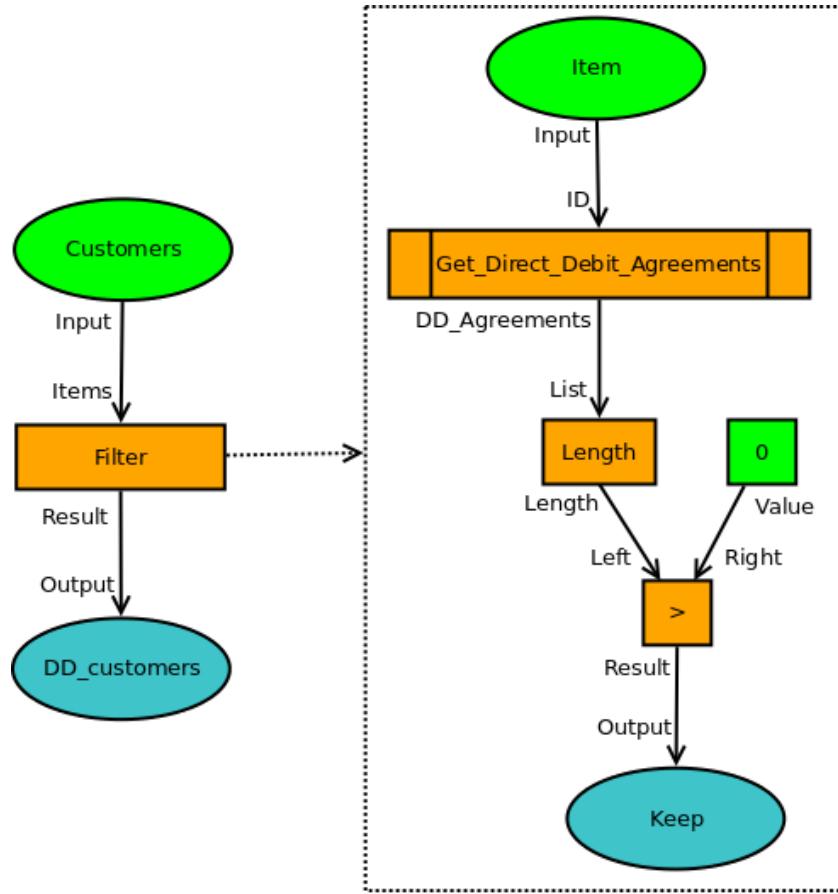


FIGURE 3.6: VOL Data Flow Filter by Direct Debit

The data flows are useful for implementing read-transform-write patterns by orchestrating service invocations and data transformations. However, data flows cannot be used to implement workflows, this is one of the things the control flow is particularly useful for.

3.1.3 Control Flow

A control flow is the entry of all VOL applications. A control flow distinguishes from a data flow by visualizing the order in which units are executed rather than specifying the data dependencies between units.

Token	Description
	Entry of control flow.
	End of control flow.

	Raise an error.
	Execute data flow
	Execute sub control flow.
	Control flow branch based on condition.
	Execute environment interaction.

TABLE 3.1: VOL Control Flow Units

The possible control flow units are illustrated in table 3.1 alongside a short description of its semantics. Each control unit is by default an ACID transaction.

All control flows must contain exactly one **Start** unit, the **Start** unit is the beginning of the control flow.

All control flows must have at least one **Stop** unit to indicate the end of the control flow. It is, however, possible that a control flow has multiple **Stop** units if the control flow contains branches.

The **Fail** unit is used when the control flow is unable to continue a normal path of execution and the control flow is forced to fail with an error of type **Error**.

The **Dataflow reference** unit is used to execute a data flow. Since all control flow units, except from the **Sub control flow reference**, are ACID transactions, if an error happens in the middle of a data flow, then none of the updates from within the data flow will be committed. Later we will see how to handle errors using the special **Error** branch which is visible to control flow units that can fail.

The **Sub control flow reference** unit is used to execute a control flow.

The **Decision** unit is used for branching to one branch if a condition is true and branching to another branch if the condition is false.

The **Interaction reference** unit is used for service invocations which cannot be inside an ACID transaction. For example, the **Interaction reference** is commonly used for user interactions which require some application user to provide input. Another use of the **Interaction reference** is for a service for sending emails. This unit can have multiple branches. For example, an **Interaction reference**, which shows a screen with two buttons, commonly has two branches. One branch is taken if one button is pressed, and the other branch is taken if the other button is pressed.

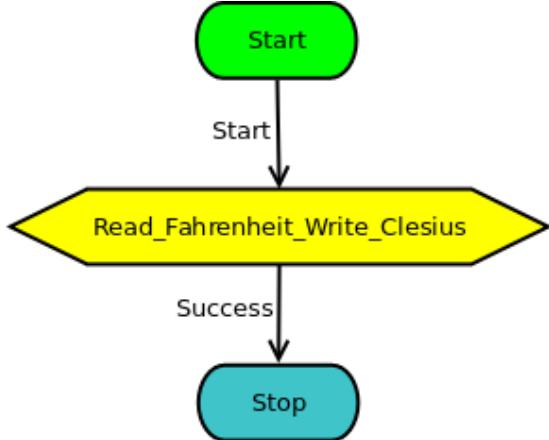


FIGURE 3.7: VOL Control Flow Read Fahrenheit Write Celsius

Figure 3.7 shows a simple control flow, which uses the data flow in figure 3.4, assuming that data flow is called **Read_Fahrenheit_Write_Celsius**. The only thing the control flow does is to start, execute the data flow and then end (return).

Notice that all control flow branches are labeled to disambiguate between branches when a control flow unit has multiple outgoing branches.

We can use output from one control flow unit as input to another control flow unit. This is necessary because many control flow units have input or output. For example, the **Decision** unit has a Boolean type input, and there are many **Dataflow reference** and **Interaction reference** units which have input or output. How to model this flow of data between control flow units is left implementation defined.

The reason for letting the modelling of data flow between control flow units implementation defined is because we suspect that visualizing data flow in the control flow graph will introduce too many crossing edges.

Figure 3.8 demonstrates a control flow for implementing a workflow for letting a Netbank user apply for a loan online. The data flows and interactions, which are used in the example, are explained together with the control flow in the following enumeration:

1. Data flow `Apply_for_loan_message` is executed for obtaining a message which is used as input for 2.

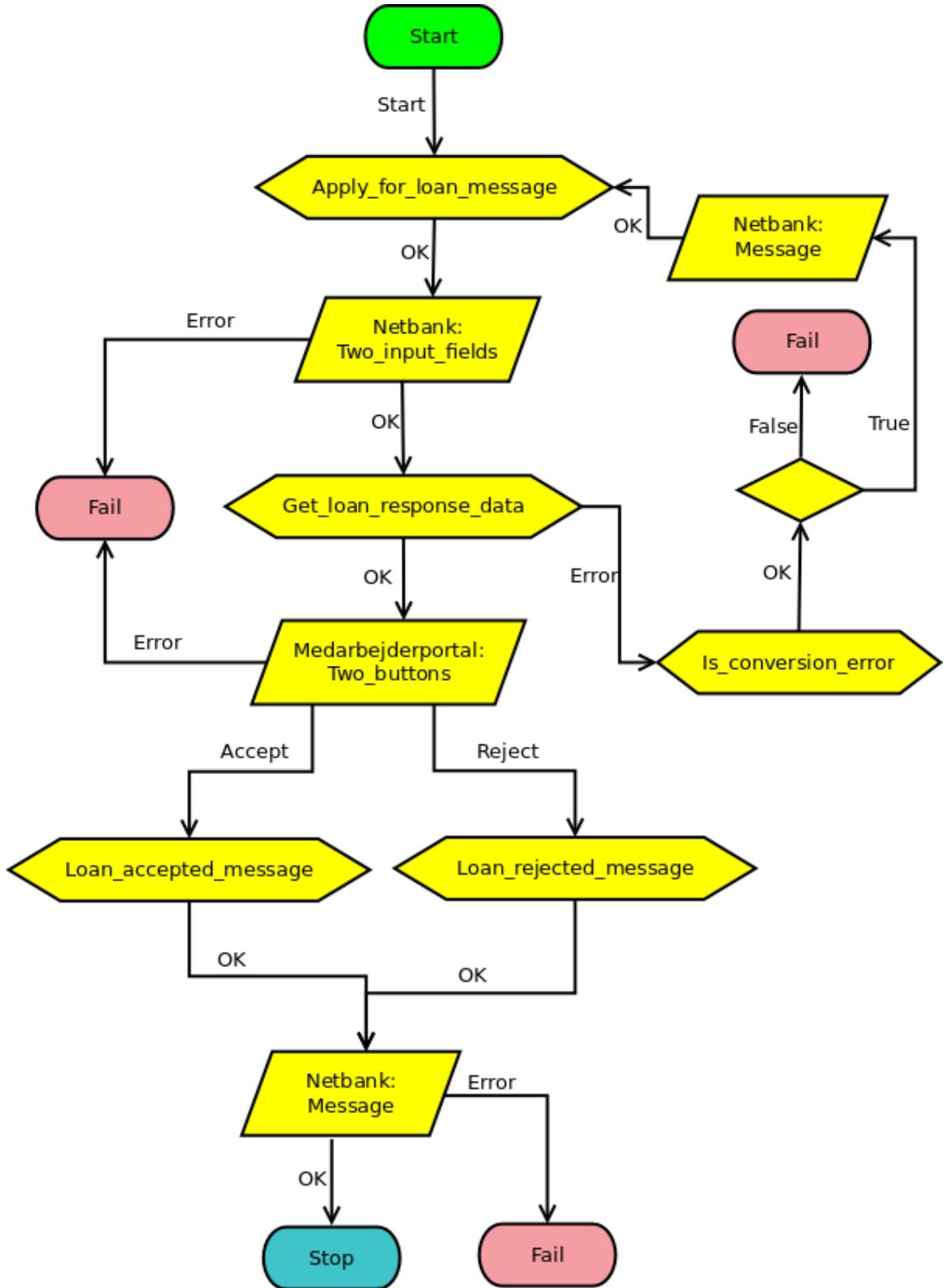


FIGURE 3.8: VOL Control Flow Apply for Loan

2. The Netbank interaction called `Two_input_fields` is executed by passing the message from 1 as input. This interaction shows the message to the Netbank user who started the program. The message asks the user to enter how much he wishes to

loan and enter a loan duration. If this interaction fails, the **Error** branch is taken and the Netbank user is shown an error message, and the application terminates.

3. Data flow `Get_loan_response_data` is executed. It is used to construct data which will be used as input for 4. It will try to convert the data, the Netbank user entered in step 2, to numbers. Since this conversion might fail, there is an **Error** branch. Although we know that the only way that the data flow can fail is by a conversion error, which we can recover from, the code is better prepared for modifications if we validate that the error code corresponds to a conversion error before recovering. This is what we use the **Decision** unit for. Unexpectedly, if the error is not a conversion error, then the application terminates and gives an error message to the Netbank user. If the error is a conversion error, then the Netbank user is told about the conversion problem, and the application is essentially restarted by going back to step 1. Either way, if data flow `Get_loan_response_data` does not fail, then it will output data which will be used in the next step.
4. Medarbejderportal interaction `Two_buttons` is executed by passing the data from 3. The data is used for providing messages. The interaction waits for a banker to open it. Once opened, the interaction shows the banker a message about the Netbank user applying for a loan and asks the banker to accept or reject the loan. If the interaction fails, then the application terminates by showing the Netbank user an error message.
5. If the loan is accepted by the banker, data flow `Loan_accepted_message` is executed to provide input message for 6, where the message says that the loan is accepted. If the loan is not accepted, data flow `Loan_rejected_message` is executed for providing the input message for 6, where the message says that the loan is rejected.
6. Netbank interaction `Message` is executed, given input from 5. The interaction shows the message to the Netbank user. If the interaction fails, the application terminates and shows the Netbank user an error message.

The control flow in figure 3.8 can be extended, for example, to invoke necessary services to register the loan in the system, if the loan is accepted.

As we can see in the figure, the VOL control flow design makes it useful for implementing workflows and handling errors.

3.2 Evaluation

In this section we will evaluate the design of VOL by comparing VOL with other languages and by validating whether VOL satisfies the design requirements from section 2.2.3.

3.2.1 Comparisons with Other Languages

By comparing VOL with other languages we can establish advantages and disadvantages of the design.

The following sections contain comparisons of VOL with the languages from section 2.3.

3.2.1.1 Comparison with BPMN

Here we will investigate the similarities and differences between the VOL control flow and BPMN.

Table 3.2 illustrates that the VOL control flow units and edges all have corresponding components in BPMN. The **VOL** column contains a VOL control flow notion and the **BPMN** column contains BPMN components used for modelling the corresponding behaviour.

VOL	BPMN
Dataflow reference unit	Call Activity [12, p. 183]
Interaction reference unit	Task [12, p. 156] possibly followed by Exclusive Gateways [12, p. 290]
Sub control flow reference unit	Call Activity [12, p. 183]
Decision unit	Exclusive Gateway [12, p. 290]
Start unit	None Start Event [12, p. 240-242]
Stop unit	None End Event [12, p. 247]
Fail unit	Error End Event [12, p. 247]
Error branch	BPMN Intermediate Error Event [12, p. 255]
Edges	BPMN sequence flow [12, p. 97]

TABLE 3.2: VOL Control Flow Notions and Corresponding BPMN Components

The data flow between VOL control flow units is implementation defined. In BPMN we can model data flow between components by using Data Objects with Data associations [12, p. 222]. We do not expect a VOL implementation to model data flow between control flow units like BPMN, since we expect that it will introduce too many crossing edges.

BPMN has an impressive number of different types of components. This is necessary since BPMN aims to support modelling communication between business processes using collaboration diagrams and choreography diagrams. Communication between business

processes is not a goal of VOL, as we can see by investigating the user requirements in section 2.2.

Since the VOL control flow has fewer components than BPMN, and since the VOL control flow is similar to the BPMN process diagram, we expect the VOL control flow to be easy and intuitive for bankers with BPMN experience.

3.2.1.2 Comparison with Oracle BPEL Process Manager

Oracle BPEL Process Manager is more control flow oriented than VOL. In Oracle BPEL Process Manager all service invocations are happening inside the control flow, in contrast to VOL where service invocations happen in the data flow.

The advantages of having service invocations in control flow, as in BPEL, are:

- Exact control of when services are invoked.
- One service can both return data and update data. This is not desirable if service invocations are in data flows. If service invocations are in data flows, then all services must either return data or update data, not both. This is so because the order of evaluation of data flows is often non-deterministic. If sinks are always evaluated last, then these rules ensure that all data flow results are deterministic even though the evaluation order is not always entirely deterministic.

The advantages of having service invocations in data flow, as in VOL, are:

- Data flow services can often be automatically invoked in parallel.
- When data flows are automatically ACID transactions then manual ACID transaction handling can be avoided.

Together with SDC we have estimated that adding services which either return data or update data, but not both, is realistic in VOL, since this is what the majority of services do anyway. Besides, when adding a service which both returns data and updates data, it is possible to simply ignore the returned data.

When we say a service which both returns data and updates data, we do not mean a service which returns an error code and updates data. This is allowed because the returned error code will not be visible in the data flow. The error code will be checked, and if it is an error, then the data flow is rolled back, and the **Error** branch in the control flow is taken.

The safe choice is to have service invocations in control flows and add manual transaction handling like Compensation Handlers in BPEL [13, p. 118] or like Transaction Subprocesses in BPMN [12, p. 178]. However, the temptation of avoiding these transaction

constructs has led us to having services in data flows. Like this we hope to avoid the type of problems which are related to users misusing or forgetting to use ACID transaction handling.

A GUI feature of Oracle BPEL Process Manager, which is convenient, is that the edges of the control flow are automatically generated when components are dragged and dropped on the control flow. This ensures a consistent code formatting.

3.2.1.3 Comparison with LabVIEW

Like we used BPMN to evaluate the VOL control flow, we will use LabVIEW to evaluate the VOL data flow.

The VOL data flow is similar to the LabVIEW data flow except that VOL uses **Functional transformations** where LabVIEW uses loops.

Figure 2.4 from section 2.3.3 shows how a gray rectangle containing data flow components is used to represent a while loop in LabVIEW. In LabVIEW we can configure while loops and for-each loops to accumulate results. Such an accumulating loop can, for example, be implemented in VOL by using an **Accumulate** transformation with an associated data flow.

Since the VOL data flow is syntactically and semantically similar to the LabVIEW data flow, we know that we are using a well tested data flow design.

LabVIEW is also interesting from an interaction design perspective. Since LabVIEW has been around for a long time, we expect LabVIEW to have an optimized development speed. While optimizing a VOL implementation's interaction design, it will be helpful to study LabVIEW.

3.2.1.4 Comparison with SSIS

The overall structure of SSIS and VOL applications is the same. They both have a control flow graph as entry point, and they both use data flows for performing read-transform-write operations.

The main differences between SSIS and VOL lie inside the data flow parts of the languages. In SSIS, tables of data are read from data sources, transformed and written. The VOL data flow has support for more than just tables of data. This is necessary to support the types of data that web services require. SSIS does support accessing web

services, but web services are invoked from the control flow rather than from the data flow.

As the SSIS data flow only supports tables of data, it is often necessary to manually type text expressions which use the built-in expression language functions to configure data flow components [14]. In VOL and LabVIEW data flow expressions are visually constructed.

The benefit of manually typing text expressions is that big expressions are faster to type and easier to read than big visual data flow expressions. The reason VOL uses visual data flow expressions is to have a consistent data flow design similar to the design of LabVIEW.

The control flows of SSIS and VOL are largely similar. Service invocations in the SSIS control flow can be used like the VOL [Interaction reference](#) control flow unit.

Most of the user stories from section 2.2.2 request a read-transform-write pattern which is what SSIS is often used for. By designing VOL with the same overall structure as SSIS with control flows and data flows, we expect the design to be well suited for most of the user stories.

3.2.2 Evaluation with Requirements

Here we use the design requirements from section 2.2.3 to evaluate VOL.

Each row of table 3.3 contains a list of requirements in column **Requirements** and a description of how the requirements have been fulfilled in column **Fulfillment description**.

Requirements	Fulfillment description
DR1	We hope that bankers recognize the similarities between the VOL control flow and BPMN, so they will find it easy to learn the VOL control flow. We do not expect that the data flow is similar to anything bankers have previously worked with, by designing the VOL data flow like LabVIEW we hope that it will not be a problem.
DR2	Since VOL is visual, this requirement is fulfilled.

DR3	When it was decided to have service invocations in the data flow, we were thinking about this requirement. Since all data flows are ACID transactions, we hope to decrease the number of problems related to data inconsistencies, compared with manual ACID transaction handling like in BPMN and BPEL.
DR4, DR13, DR14, DR17, DR18, DR19, DR20, DR21, DR23	SDC services and external services can be added to the system and used in the VOL data flows.
DR5	The VOL control flow has good support for workflows.
DR6	The VOL data flows are by default ACID transactions.
DR7	In the VOL control flow error handling is supported with the Error branch.
DR8	The VOL control flows and data flows are reusable components.
DR9, DR10	If it is possible to add Interaction reference units which support TDC scale and Follow-Up Popup interactions, then we expect to be capable of implementing TDC scale and Follow-Up Popup applications which are started by some event. If it is not possible to add these Interaction reference units, then it is possible to implement something identical to TDC scale and Follow-Up Popup.
DR11, DR12	We can add Interaction reference units which support sending emails and SMSs.
DR15	Interaction reference units for implementing user interactions with different systems can be added.
DR16	Interaction reference units with multiple branches can handle branching based on button clicks.
DR22	An Interaction reference unit which supports showing data on the screen and then allowing the data to be modified on the screen can be added to the system.
DR24, DR25	Math and transformation operations are supported by the VOL data flow.
DR26	The design of VOL does not directly support listening for events. By allowing VOL applications to be started by different systems, it is possible to have a system which listens for events and starts VOL applications when events are received.

TABLE 3.3: Design Requirement Fulfillment Descriptions

3.3 Hypotheses

The fulfillment descriptions of requirements [DR1](#) and [DR3](#) gives rise of hypotheses about the design of VOL. The following two sections will describe these hypotheses.

3.3.1 VOL is Easy for Bankers

Requirement [DR1](#) gives rise of the hypothesis that the design of VOL makes it easy to use for bankers.

In chapter [4](#) we will develop an implementation of VOL and test the hypothesis, by showing the implementation to a bank specialist from SDC in section [4.3.1](#).

3.3.2 ACID Transaction Data Flow

Requirement [DR3](#) states a hypothesis by saying: When having service invocations in data flows and requiring all data flows to be ACID transactions, we decrease the number of problems with data inconsistencies, compared with having service invocations in the control flows and manual ACID transaction handling, like Transaction Sub-processes in BPMN [\[12, p. 178\]](#) or Compensation Handlers in BPEL [\[13, p. 118\]](#).

This hypothesis can be confirmed or disproved by implementing two versions of VOL. One version with ACID data flows containing service invocations and another version with service invocations in the control flow and manual ACID transaction handling. By testing applications implemented by both versions over a sufficient period of time, we can count incidents where incorrect ACID transaction handling may cause data inconsistencies. If there is a clearly lower number of ACID transaction handling problems with ACID data flows containing service invocations, then the hypothesis is confirmed, otherwise the hypothesis is disproved.

In section [5.2.1.4](#) we will further discuss testing this hypothesis.

Chapter 4

Implementation

This chapter describes a Proof of Concept (PoC) implementation of VOL, which is called Bank Integrated Development Environment (BIDE). The design of VOL is described in section [3.1](#).

4.1 BIDE User Manual

4.1.1 Getting Started with BIDE

When BIDE is opened the user is shown the start screen in figure [4.1](#).

From the start screen the user has two ways to open a project. The user can open an existing project by clicking **File** → **Open project**, or the user can open a new project by clicking **File** → **New project**, as illustrated in figure [4.2](#).

When opening an existing project, the user is shown a file chooser he can use to find the project file he wants to open.

When opening a new project, the user is asked to select a platform, a user group and a project title. The platform corresponds to the system from where the application will be started. BIDE supports two imaginary platforms called Medarbejderportal and Netbank. The user group specifies who are allowed to start the application. Two user groups are supported: Normal and Administrator. The project title is used to specify a project identifier, which will be the name of the entry (main) control flow as well.

While a project is open, BIDE contains a project tree explorer which can be used to open control flows or data flows. Figure [4.3](#) shows an example of a project tree explorer.

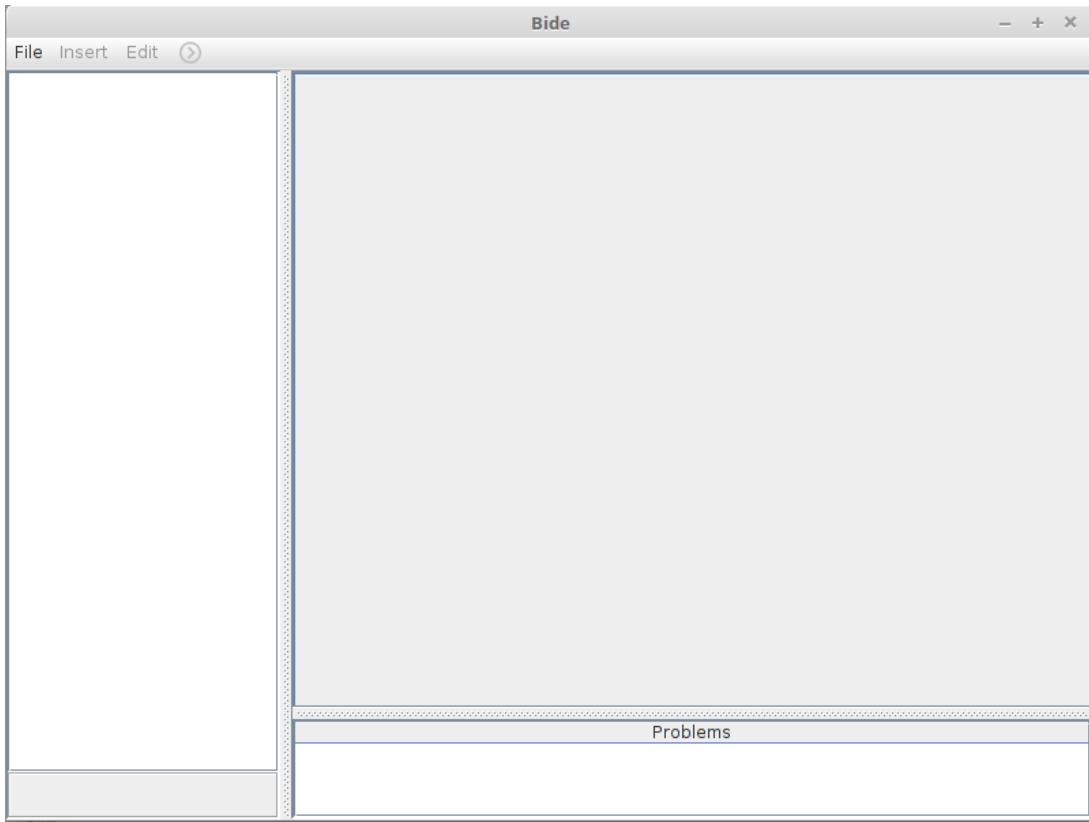


FIGURE 4.1: BIDE Start Screen

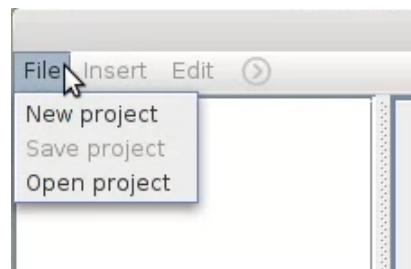


FIGURE 4.2: BIDE File Menu

By clicking a node starting with a green **C** the user opens a control flow, and by clicking a node starting with a blue **D** the user opens a data flow.

When a project tree explorer control flow node or data flow node is clicked, then a tab is created, and the control flow or data flow is shown in the main window. Notice that the main window is the big gray rectangle in figure 4.1. Figure 4.4 illustrates the main window showing a control flow, where there are 3 tabs, and the control flow tab `Apply_for_loan` is selected.

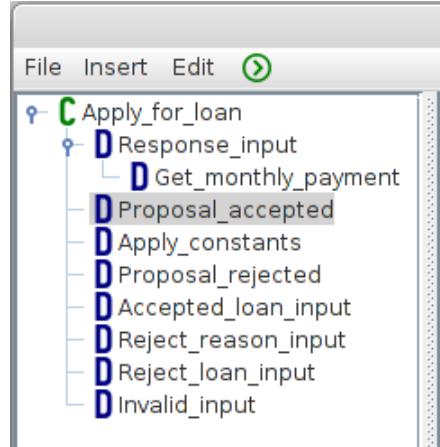


FIGURE 4.3: BIDE Project Explorer

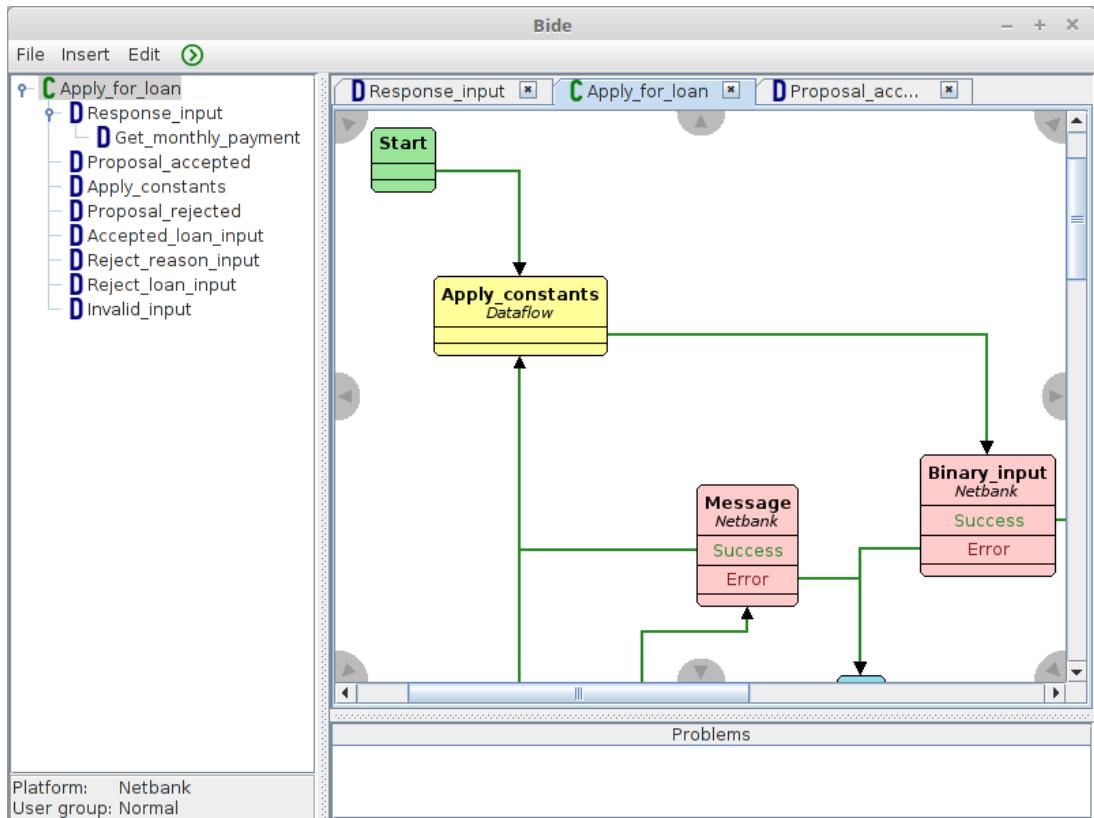


FIGURE 4.4: BIDE Multiple Tabs

4.1.2 Control Flow

While a control flow tab (a tab starting with a green **C**) is selected, the user can program the corresponding control flow. Control flows are programmed by inserting and dragging and dropping control flow units, dragging edges and specifying data mappings.

Figure 4.5 illustrates the control flow units. The units titled **Start**, **Stop** and **Fail** correspond to the **Start**, **Stop** and **Fail** units described in section 3.1.3. The unit titled **If**

corresponds to the `Decision` unit in section 3.1.3. The unit with the `Dataflow_Example` title corresponds to a `Dataflow reference` unit in section 3.1.3. The unit titled `Message` corresponds to an `Interaction reference` unit in section 3.1.3. BIDE does not have anything corresponding to the `Sub control flow reference` unit in section 3.1.3.

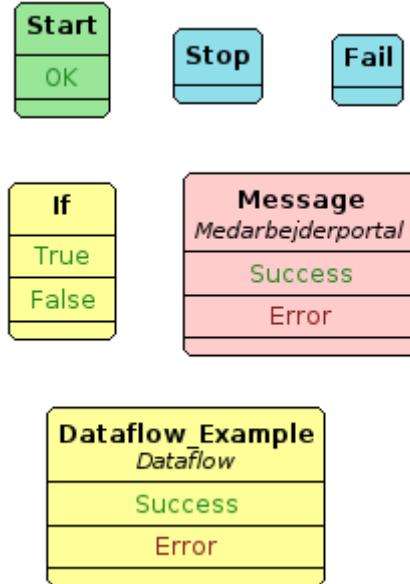


FIGURE 4.5: BIDE Control Flow Units

Whether a Data flow unit has an `Error` branch or not, is determined by whether the data flow can fail.

When inserting an Interaction unit, the user is asked to select which platform the interaction should be on and select which type of interaction he wants. In figure 4.5 we can see that there is a `Medarbejderportal` interaction of type `Message`. The `Message` interaction is an interaction which shows a message to a particular user. `Medarbejderportal` also supports other interaction types. Figure 4.6 shows the `Medarbejderportal` interaction types the user can select. Netbank supports the same interaction types as `Medarbejderportal`.

- The `Message` interaction shows a message to a particular user.
- The `Table` interaction shows a table of strings to a particular user.
- The `Unary_input` interaction prompts a particular user for input using an input field.
- The `Binary_input` interaction prompts a particular user for input using two input fields.
- The `Binary_button` interaction asks a particular user to press one of two buttons.

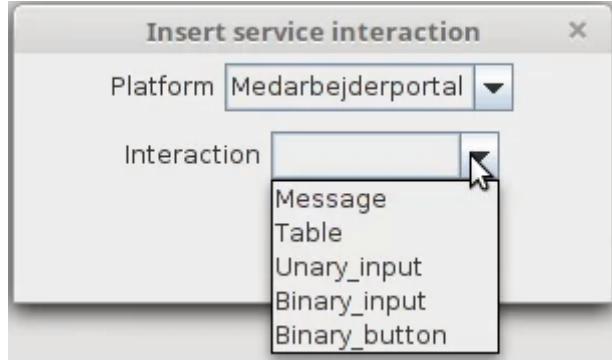


FIGURE 4.6: BIDE Control Flow Interaction Types

By positioning the mouse cursor over the left or right side of Interaction units, Data flow units, If units or Start units the user can drag control flow edges to connect units. A round marker is shown when the mouse cursor is located where the user can start dragging a control flow edge. Which side the user wants to start the control flow edge from is optional.

Figure 4.7 shows an example of a marker for starting a control flow edge from the right side of the **Success** branch of a data flow. When a marker is shown, the user can drag the mouse cursor over a control flow unit and release to create an edge to that unit.

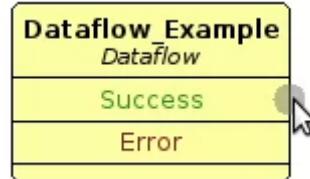


FIGURE 4.7: BIDE Control Flow Edge Start Marker

Figures 4.8a-c illustrate the creation of a control flow edge from the Start unit to a Stop unit. Figure 4.8a shows the marker for starting an edge from the left of the Start unit, the user can press the mouse to start dragging the edge. Figure 4.8b shows the edge being dragged. Figure 4.8c shows the edge dragged to the Stop unit, the user can release the mouse to create the edge.

Whether an edge starts from the left or right does not change the semantics, it is only to allow users to select side to improve readability of the graph.

When creating a control flow edge which ends at a unit which needs input, then the user is shown a prompt window to select the input from predecessor control flow units. Figure 4.9 shows the prompt, in the bottom of the figure, right after the user released the edge from the **Get_message** data flow to the **Message** interaction. Notice that the edge is red because the input for the interaction has not been specified yet.

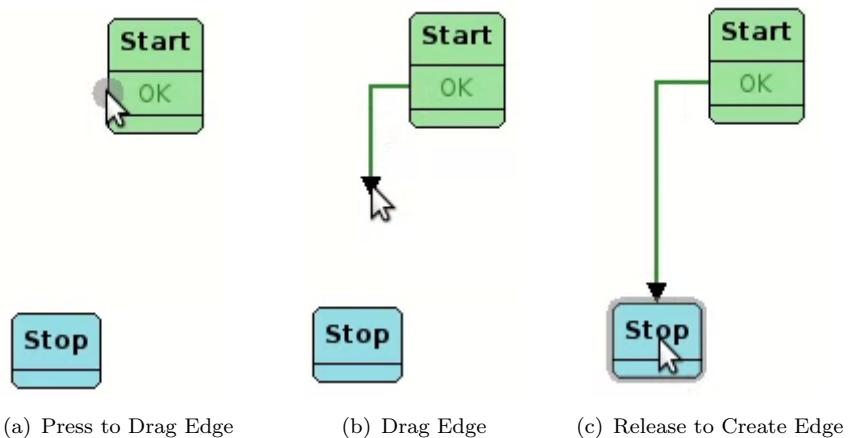


FIGURE 4.8: BIDE Control Flow Edge Creation

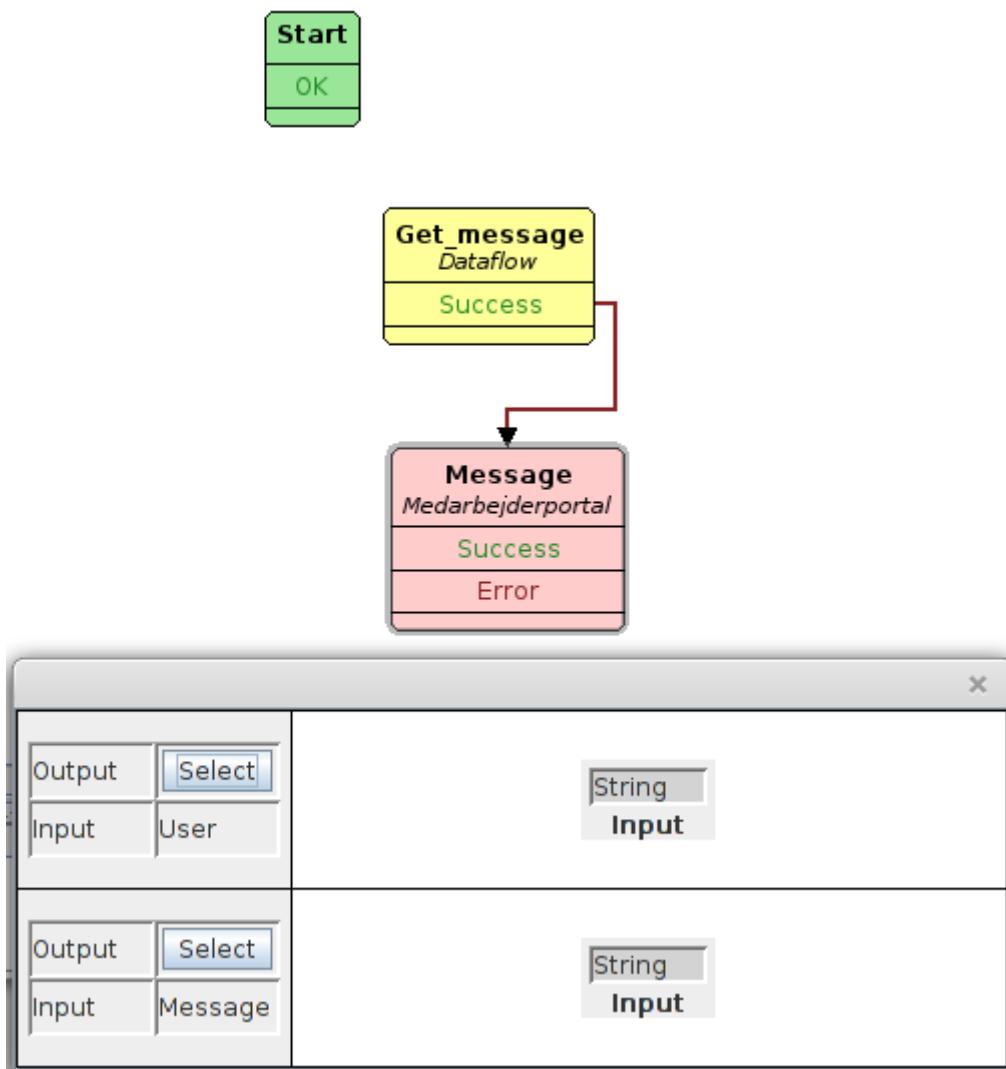


FIGURE 4.9: BIDE Control Flow Input Prompt

If the user clicks the **Select** button in the lower half of the prompt window, then he

can select the control flow unit to provide the message. Figure 4.10 illustrates how it looks when the user hovers the mouse over the `Get_message` data flow after the `Select` button has been clicked.

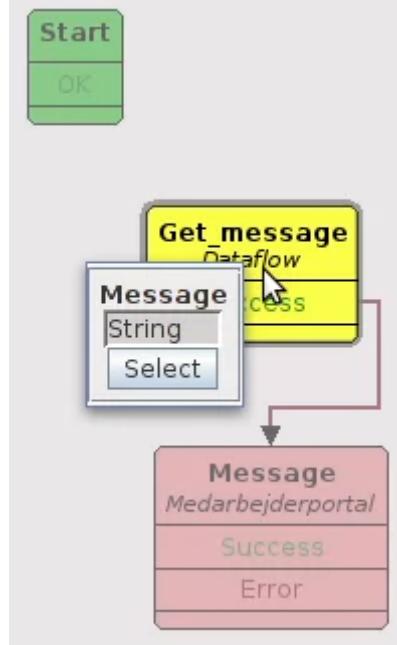


FIGURE 4.10: BIDE Control Flow Select Input

The figure illustrates that the `Get_message` data flow has a single output called `Message` of type `String`. Notice that the Start unit and the `Message` interaction are grayed out. This is because they are not predecessors of the `Message` interaction, hence they cannot provide input for it.

After the user clicks on the `Select` button in figure 4.10, the user is shown the prompt in figure 4.11.

The bottom row of the prompt contains a data mapping which shows that the message input for the `Message` interaction is now connected, see section 4.1.4 for more about data mappings.

The top row of the prompt shows that a user input is required before all the inputs for the `Message` interaction are connected. The user input is used to specify which Medarbejderportal user to receive the message. The Start unit has one output which can be used to provide the user input. The output of the Start unit is the username of the user who started the application. Note that this application is started from Medarbejderportal, so the username from the Start unit is a Medarbejderportal username, which is what the `Message` interaction's user input needs.

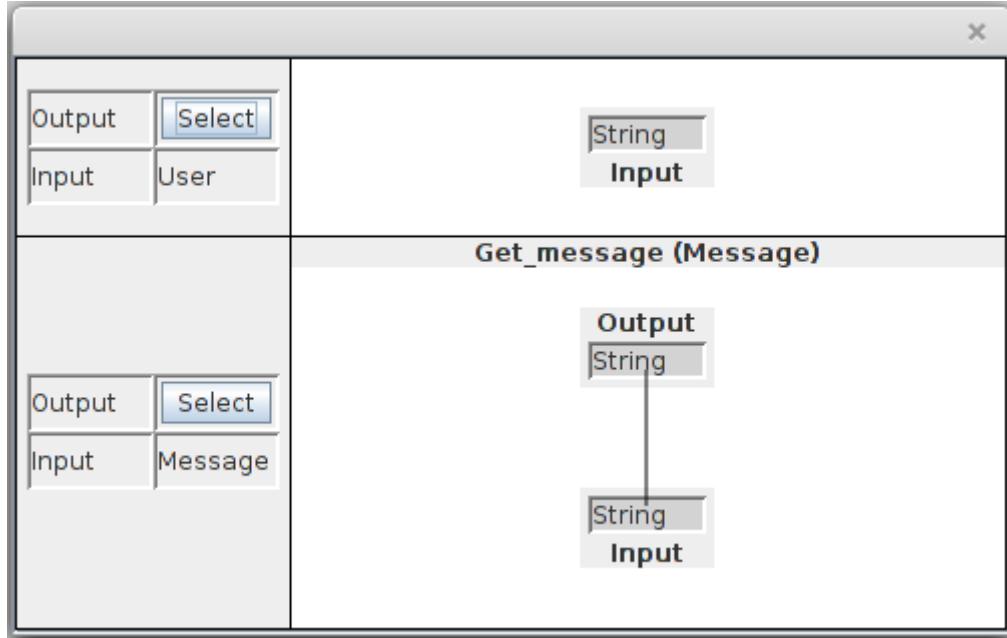


FIGURE 4.11: BIDE Control Flow Input Prompt 2

Currently, it is not possible to connect the output of the Start unit to the Message interaction, because the Start unit is not a predecessor of the Message interaction. In order to make the Start unit predecessor of the Message interaction, the prompt must be closed such that an edge from the Start unit to the Get.message data flow can be created. Figure 4.12 illustrates how the control flow graph looks after doing that.

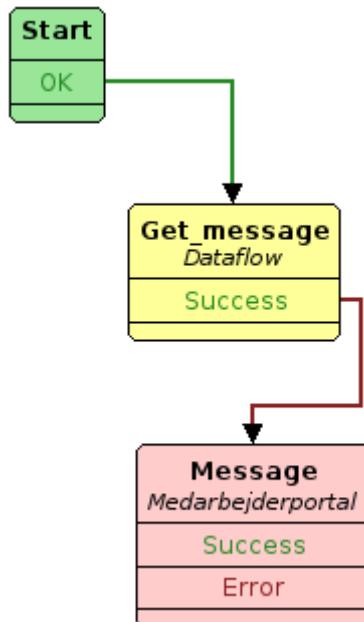


FIGURE 4.12: BIDE Control Flow Graph with Start Connected

By double clicking the red edge from the `Get_Message` data flow to the `Message` interaction the user is shown the same prompt as before, shown in figure 4.11. But this time the Start unit is predecessor of the `Message` interaction, such that the Medarbejderportal username output from the Start unit can be used as the last input for the `Message` interaction.

Figure 4.13 illustrates selecting the output from the Start unit as input for the `Message` interaction.

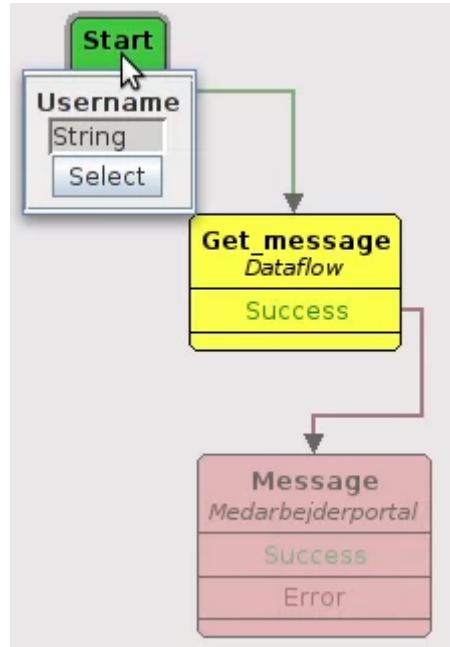


FIGURE 4.13: BIDE Control Flow Select Input 2

Pressing the `Select` button in figure 4.13 will make the edge from the `Get_message` data flow to the `Message` interaction change color to green, since the input for the `Message` interaction is now specified. Figure 4.14 illustrates how this looks.

4.1.3 Data Flow

When a data flow tab with a blue **D** is selected, the user can program the corresponding data flow. Data flows are programmed by inserting and dragging and dropping Data flow units, dragging edges between units and specifying data mappings.

BIDE does not have data flow services. Extending BIDE by adding data flow services is just a matter of time.

Except from the `Service source`, `Service transformation` and `Service sink` units, the data flow in BIDE has units corresponding to all the units from section 3.1.2.

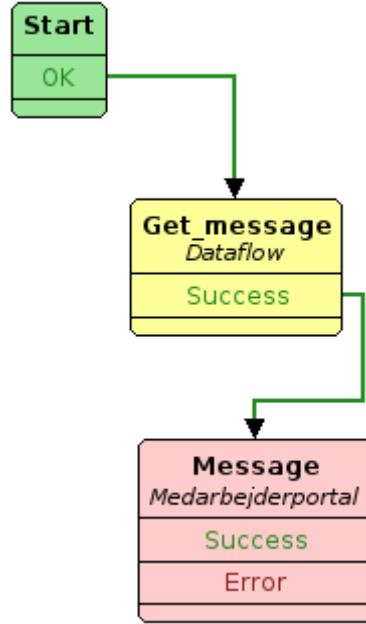


FIGURE 4.14: BIDE Control Flow All Input Selected

BIDE has data flow units for performing various operations. BIDE has arithmetic operations, string operations, comparison operations, list operations, boolean operations and more. A sample of data flow units is shown in figure 4.15.

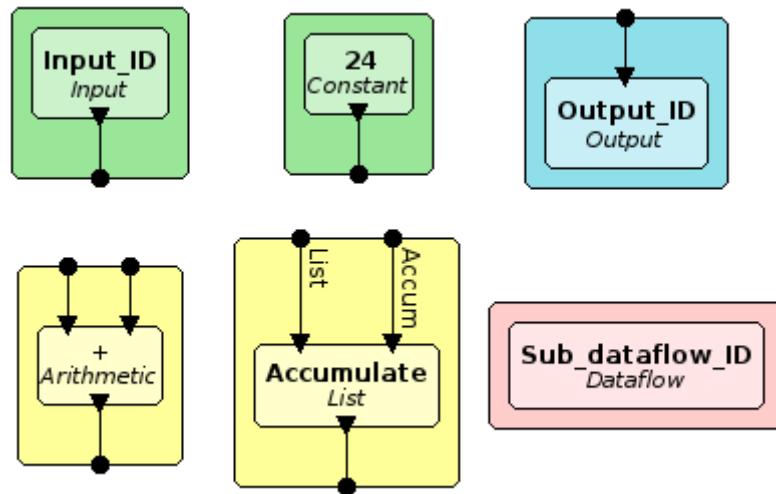


FIGURE 4.15: BIDE Data Flow Units

The subtitle with *italic* font identifies the kind of each unit in the figure.

1. The unit titled **Input_ID** is a data flow input.
2. The unit titled **24** is a constant.
3. The unit titled **Output_ID** is a data flow output.

4. The unit titled `+` is an arithmetic addition.
5. The unit titled `Accumulate` is a for-each loop for accumulating a result. By double clicking the unit a data flow representing the loop body is opened for modification.
6. The unit titled `Sub_dataflow_ID` is a data flow invocation of a data flow titled `Sub_dataflow_ID` with no input data and no output data.

The color of a data flow unit identifies whether it is a source (green), sink (blue), transformation (yellow) or procedure (red).

When a data flow unit has input, there are one or more black circles on top of the unit to indicate that the unit has input which must be connected to an output of another data flow unit. An input circle may be labeled to indicate which input the circle corresponds to. For example, the `Accumulate` unit in figure 4.15 indicates that the left input circle corresponds to the input identified by `List` and the right input circle corresponds to the input identified by `Accum`.

When a data flow has output, there are one or more black circles on bottom of the unit. Each output circle must be connected to at least one input circle from another data flow unit. An output circle may be labeled to indicate which output the circle corresponds to.

By placing the mouse over an input or output circle, the user can drag an edge to another circle. All edges must connect an output to an input, and cycles in the data flow graph are not allowed.

Figures 4.16a-c illustrate the creation of a data flow edge. Figure 4.16a shows the mouse over an output circle, by pressing the mouse the user can start dragging the edge. Figure 4.16b shows the mouse dragging the edge. Figure 4.16c shows the edge dragged over an input circle where the user can release the mouse to create the edge.

Figure 4.17 shows a data mapping window which is shown after the mouse has been released in figure 4.16c. When the type of the output and the type of the input are equal, the type mapping is automatically created. Section 4.1.4 gives an explanation of data mappings.

4.1.4 Data Mapping

Data Mappings are used for mapping output data to input data. Section 4.1.2 and section 4.1.3 contains examples showing where data mappings are used.

Figures 4.18a-c illustrate the usefulness of data mappings by allowing mapping a `Number` from a `Record` output to a `Number` input.

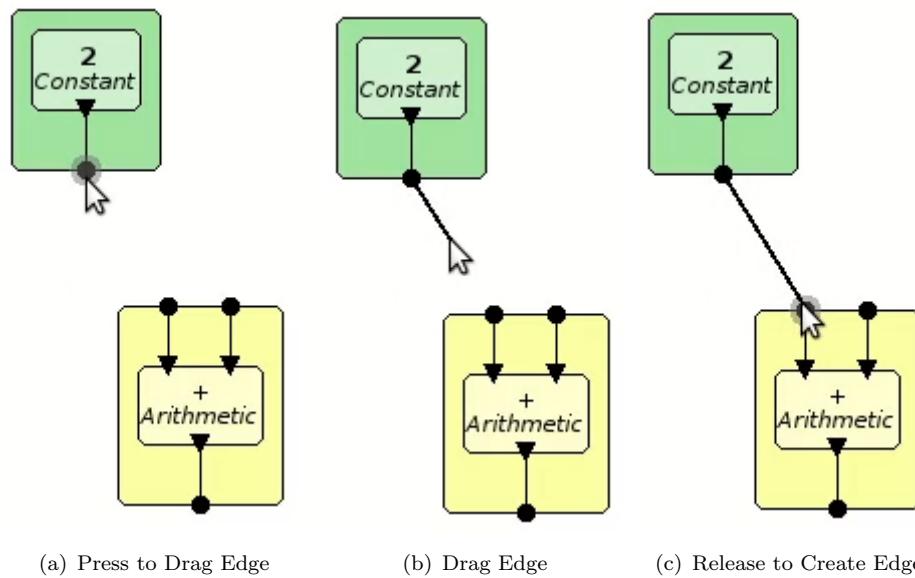


FIGURE 4.16: BIDE Data Flow Edge Creation

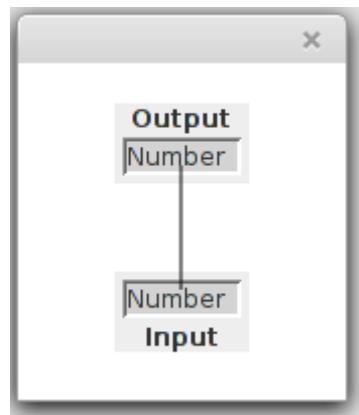


FIGURE 4.17: BIDE Data Flow Data Mapping

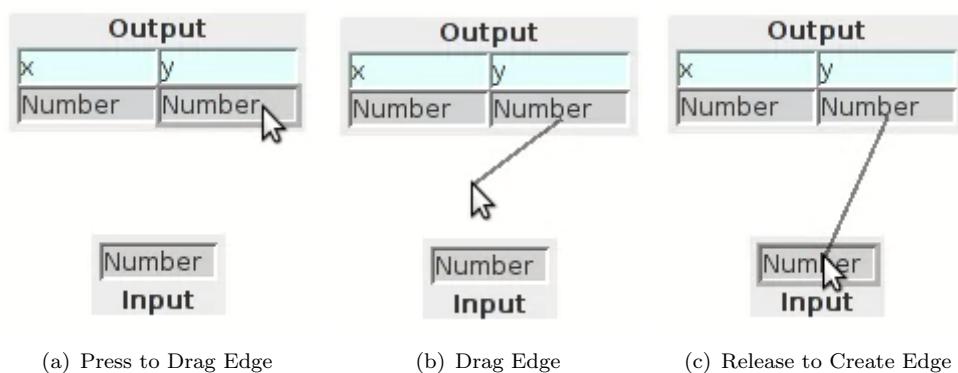


FIGURE 4.18: BIDE Data Mapping from Record

BIDE cannot know whether the user wants to map the **Number** identified by **x** or the **Number** identified by **y**, hence the user must manually perform the mapping. The figures

illustrate mapping the **Number** under the **y**. Figure 4.18a shows the user hovering the mouse over the **Number** under the **y**, the user can press the mouse to start dragging an edge. Figure 4.18b shows the user dragging the edge. 4.18c shows the user has dragged the edge over the **Number** input, the user can release the mouse to create the edge connection.

Another use of a data mapping is to map list data. Figure 4.19 shows a mapping from one type of list to another.

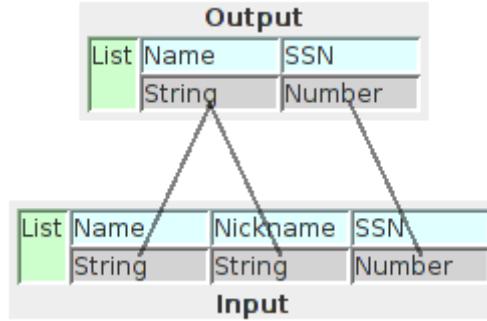


FIGURE 4.19: BIDE Data Mapping from List

The figure shows a mapping from a list of people with a name and a SSN to a list of people with a name, nickname and SSN. The figure shows that the name in the output list is used for both name and nickname in the input list, and the SSN in the output list is used as SSN in the input list.

A data mapping cannot connect different types. A data mapping can, for example, not connect a **Number** to a **String**. To implement such a conversion a Type cast data flow unit can be used.

A data mapping cannot connect two different lists to a single list. For example, figure 4.20 illustrates a data mapping which is impossible to make. This mapping is not allowed since the two input lists might not have the same length.

4.1.5 Example

4.1.5.1 Application Implementation

In this section we will implement a workflow application using BIDE. The application will be started from Netbank and ask the user who started the application to enter a Medarbejderportal username to send a greeting. After the Medarbejderportal user has seen the greeting the application will terminate.

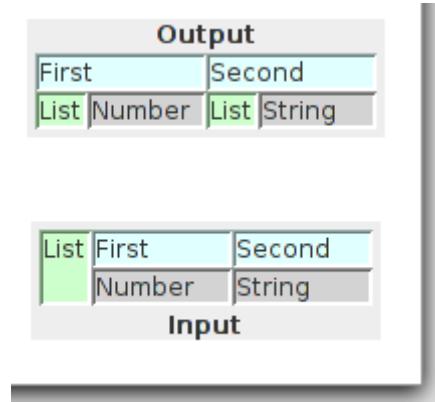


FIGURE 4.20: BIDE Data Mapping from Two Lists to One List

After opening BIDE we go to **File → New project** to open a new project. This will ask us to select platform, select user group and enter the project title. Figure 4.21 shows that we select Netbank as start platform, we select that all users can use the application, and we enter the project title: `Greet_Medarbejderportal_User`.

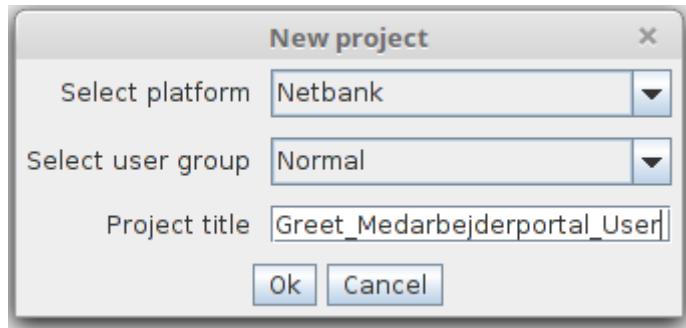


FIGURE 4.21: BIDE Example New Project Prompt

While the control flow is opened in the main window, we go to **Insert → Service interaction**. This gives us a prompt which will ask us to select platform for the interaction and select type of the interaction. Figure 4.22 shows that we select Netbank as platform and select the `Unary_input` interaction type.

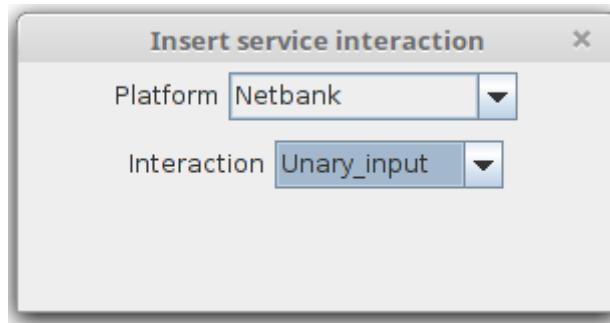


FIGURE 4.22: BIDE Example Input Interaction Prompt

In order to provide input to the `Unary_input` interaction we insert a data flow by going to **Insert → New dataflow**. Figure 4.23 shows that we give this data flow the title `Select_User_Messages`.

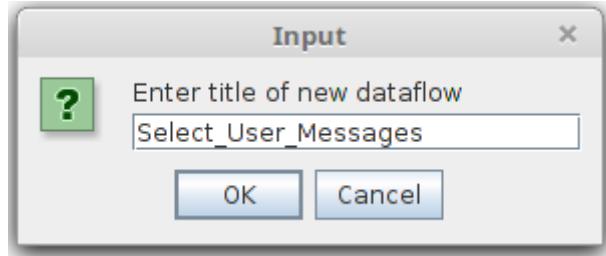


FIGURE 4.23: BIDE Example Unary Input Dataflow

At the moment the control flow looks as shown in figure 4.24.

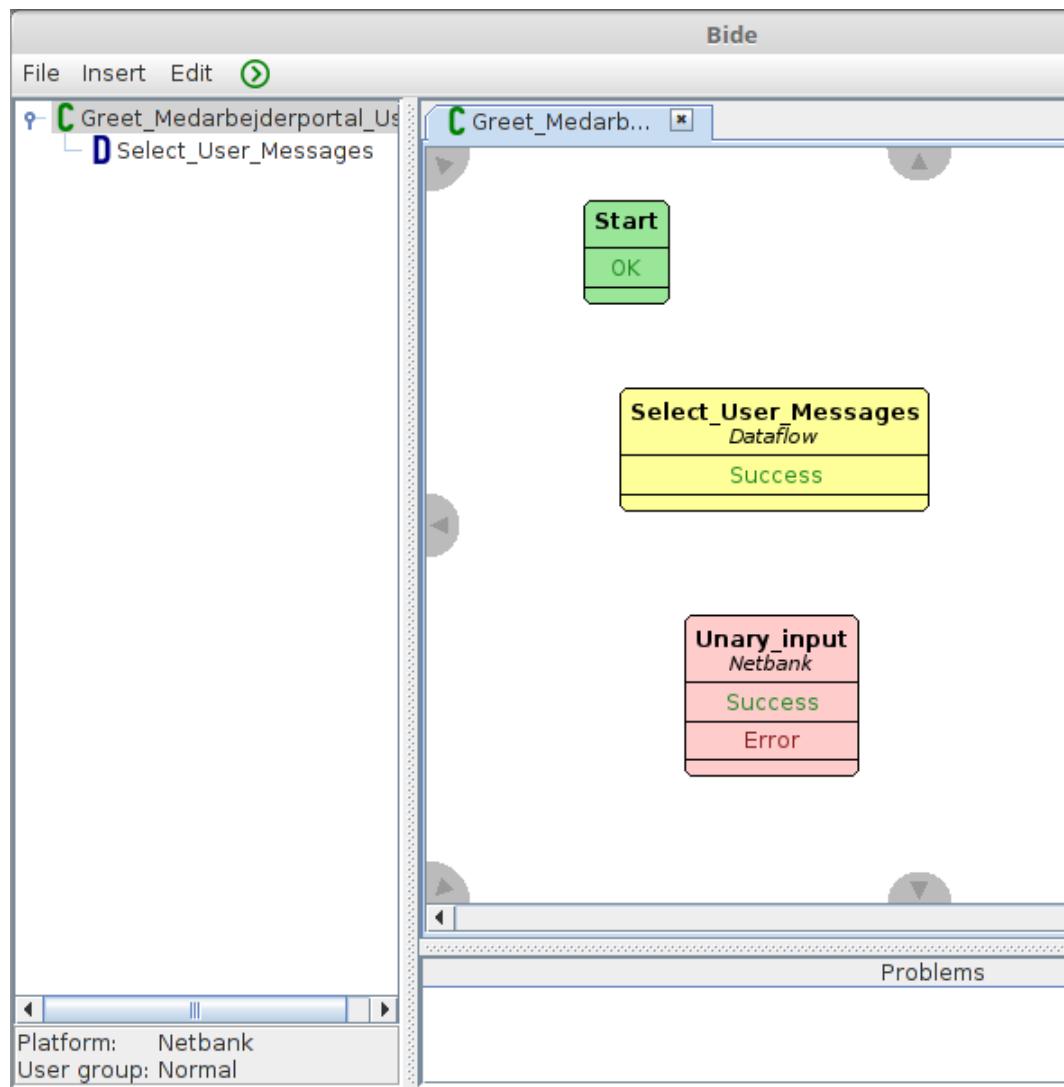


FIGURE 4.24: BIDE Example Control Flow 1

By double clicking the `Select_User_Messages` data flow in the main window or by clicking the `Select_User_Messages` node in the project tree explorer we can open the data flow. While the `Select_User_Messages` data flow is open in the main window, we go to **Insert → Insert component → Constant**. This will provide us with the window shown in figure 4.25.

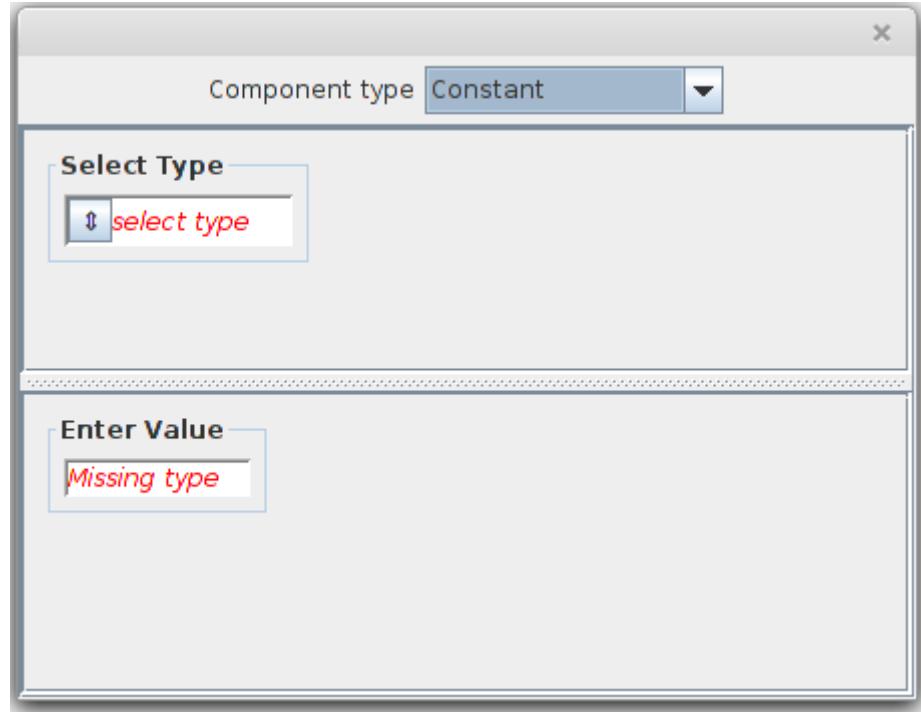


FIGURE 4.25: BIDE Example Insert Constant 1

The `Select Type` drop down box allows us to select the type of the new constant. We can select from the types shown in figure 4.26. As we can see, BIDE supports all the VOL types from section 3.1.1 except from `Date` and `Time`.

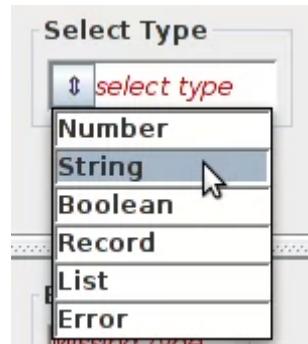


FIGURE 4.26: BIDE Example Type Selection

If we select `List`, we are asked to select the type argument as well, this is illustrated in figure 4.27a. If we select `Record`, we are asked to provide the named type arguments,

this is illustrated in figure 4.27b

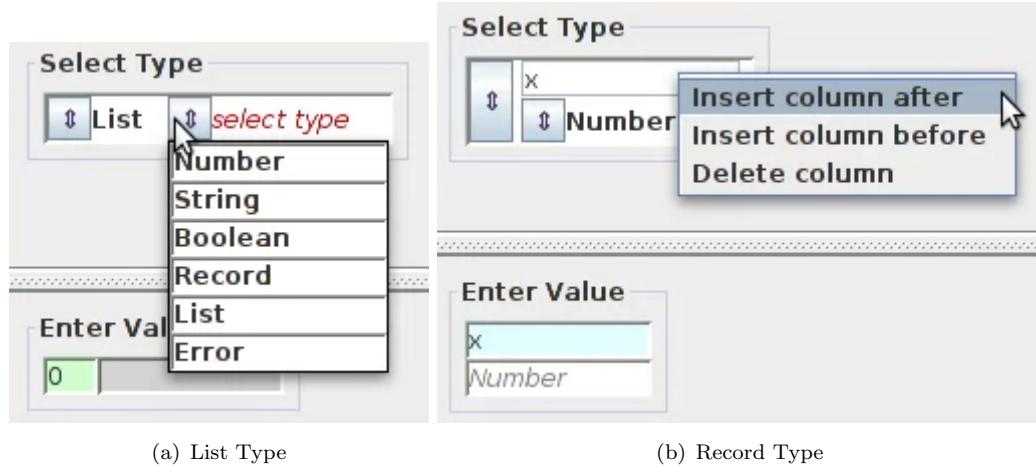


FIGURE 4.27: BIDE Example Types

In our example we only need a **String**, so that is what we will select. We will enter the value `Who would you like to greet on Medarbejderportal?`. This is illustrated in figure 4.28.

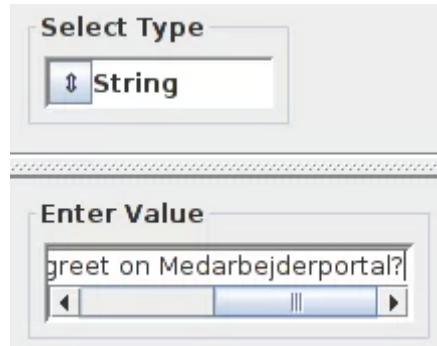


FIGURE 4.28: BIDE Example String Constant

After closing the window we can see the Constant unit in the data flow.

We would like to insert another Constant unit in the data flow. We do as before by going to **Insert → Insert component → Constant** and selecting **String** type, but this time we enter the constant value `Enter the username here`.

Now the data flow looks as shown in figure 4.29.

We want to use the two constants as input for the `Unary_input` interaction, hence we need to output the constants from the data flow. To do so we insert two Output units. To insert an Output unit we go to **Insert → Insert component → Output**, which will give us the window which is shown in figure 4.30.

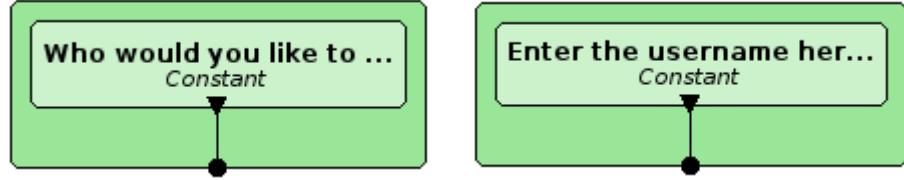


FIGURE 4.29: BIDE Example Data Flow 1

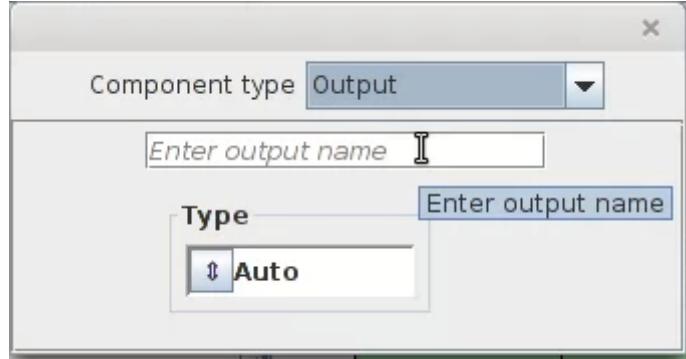


FIGURE 4.30: BIDE Example Insert Output

The Output unit supports the `Auto` type, which is a type that automatically adapts, so the `Auto` type can become any type. We will see how the `Auto` type works later. After entering the output identifier `Msg` and closing the window we get an Output unit in the data flow.

By following the same procedure to add one more Output unit with `Auto` type and identifier `Lbl`, we get the data flow which is shown in figure 4.31.

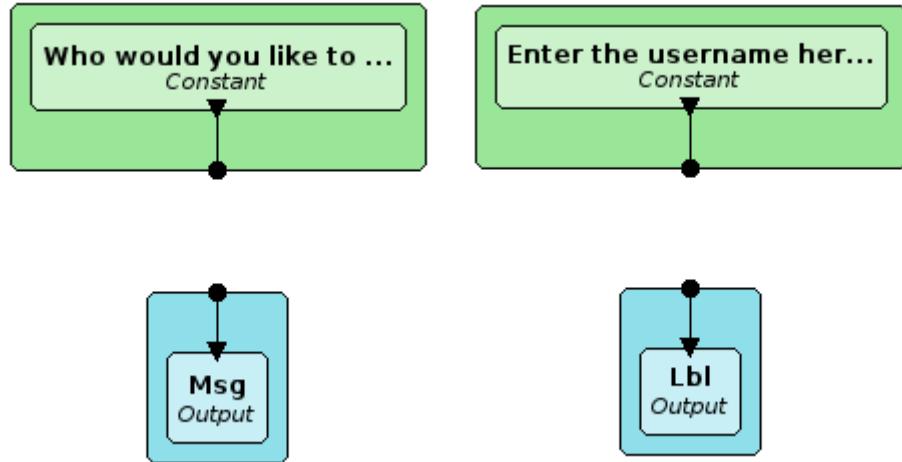


FIGURE 4.31: BIDE Example Data Flow 2

Now we will connect the left constant with the left output and connect the right constant with the right output. To do so we drag an edge from the left Constant unit's output

circle to the left Output unit's input circle. This gives us the type mapping window in figure 4.32.

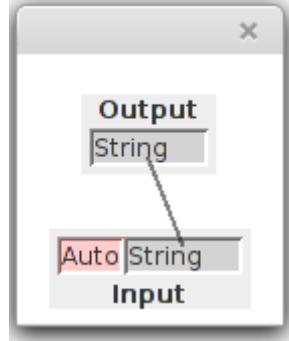


FIGURE 4.32: BIDE Example Auto Type Mapping

The figure illustrates that the Output unit's **Auto** type has automatically adapted and has converted into a **String**. The data mapping is also automatically created for us, so we can close the window.

We do the same to connect the right Constant unit's output to the right Output unit's input. This finishes the data flow, which is shown in figure 4.33.

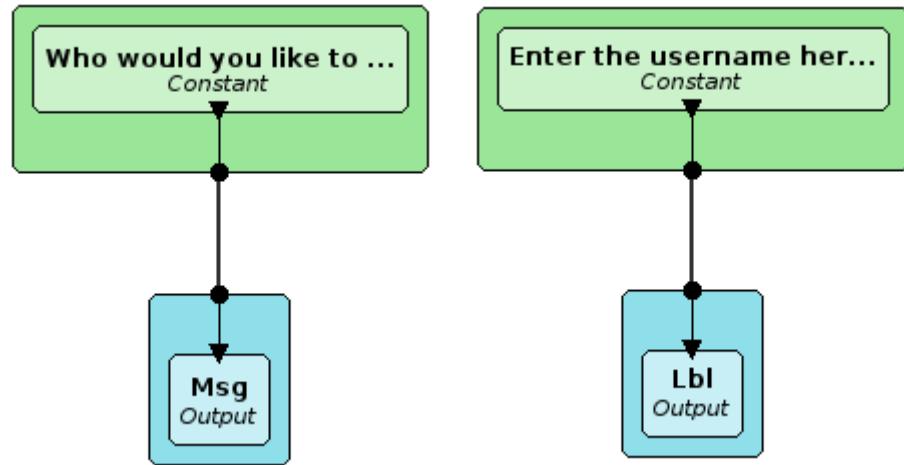


FIGURE 4.33: BIDE Example Data Flow 3 (Finished)

Now we will open the control flow in the main window and drag an edge from the Start unit to the **Select_User_Messages** data flow. Afterwards, we will drag an edge from the **Select_User_Messages** data flow to the **Unary_input** interaction. After the edge to the **Unary_input** interaction is created, we are given the window shown in figure 4.34.

It asks us to select data mappings for the **Unary_input** interaction inputs. For the **User** input we select the **Username** output from the Start unit. For the **Message** input we select the **Msg** output from the **Select_User_Messages** data flow. For the **Label** input

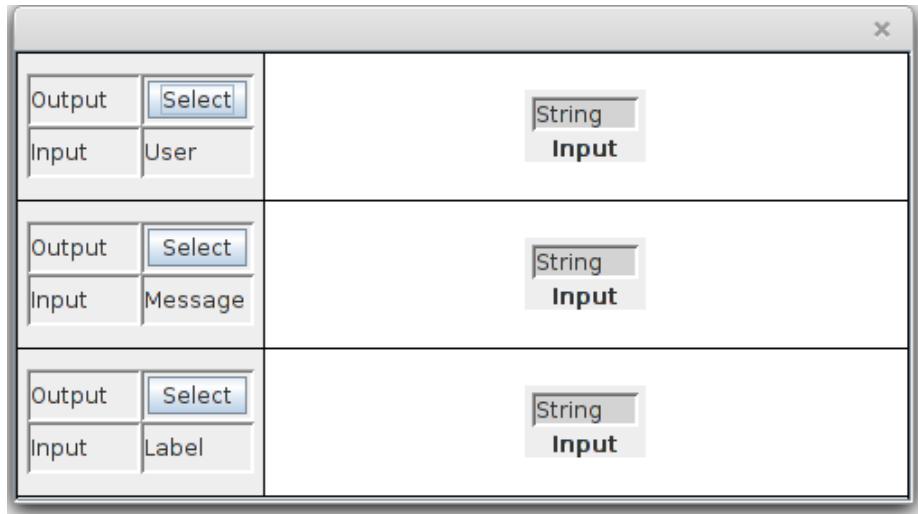


FIGURE 4.34: BIDE Example Control Flow Input Prompt

we select the `Lbl` output from the `Select_User_Messages` data flow. When all the input has been connected, the window looks as shown in figure 4.35.

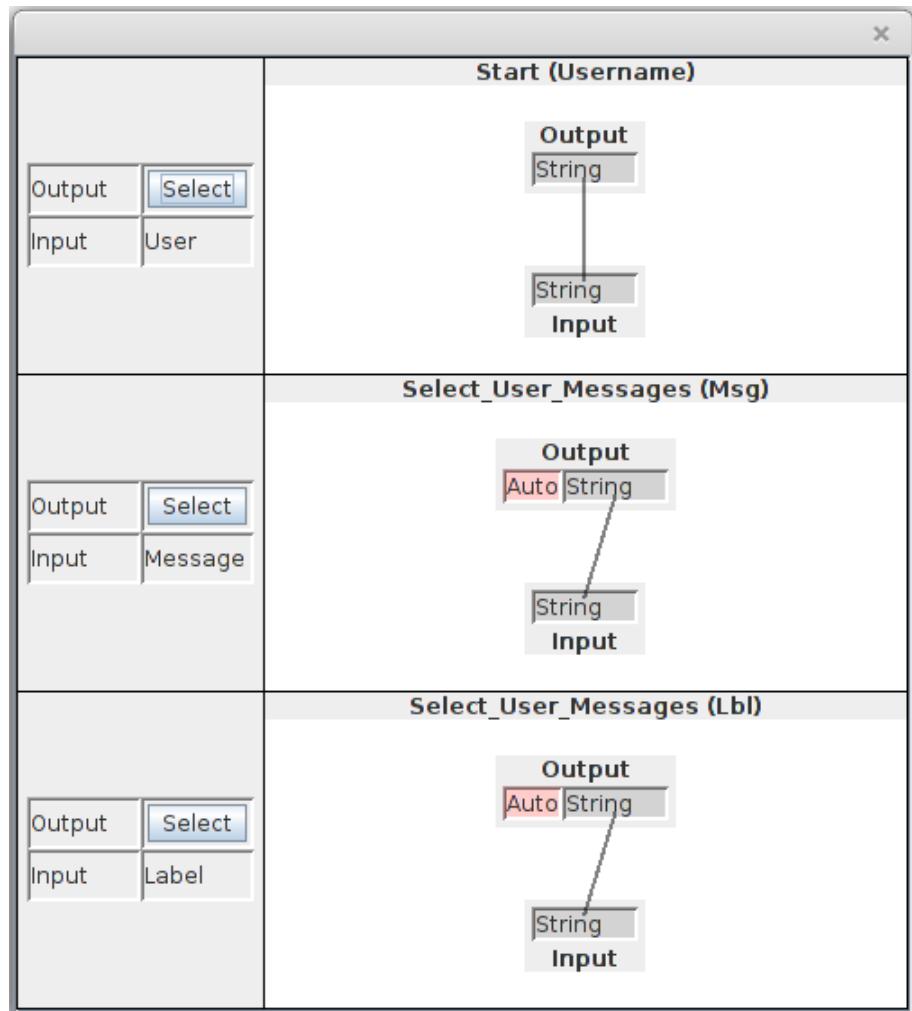


FIGURE 4.35: BIDE Example Control Flow Finished Input Prompt

After closing the window the control flow looks as shown in figure 4.36.

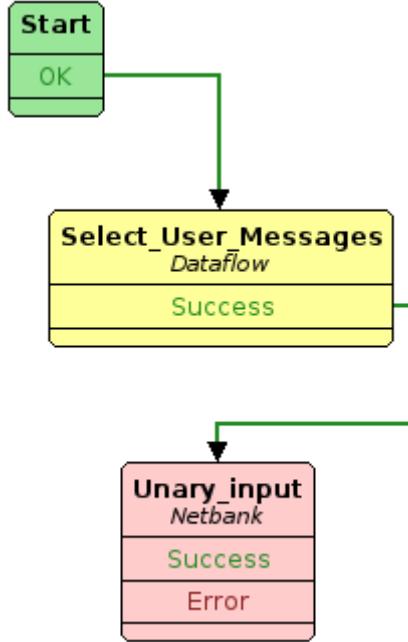


FIGURE 4.36: BIDE Example Control Flow 2

Now we insert a new data flow in the control flow by going to **Insert → New dataflow**. We give the data flow the title `Greeting_Message`.

Then we open the `Greeting_Message` data flow in the main window. To implement this data flow we insert an Input unit by going to **Insert → Insert component → Input** and giving the input the type `String` and identifier `Username`. We insert a String format unit by going to **Insert → Insert component → Functional → String → Format** and entering the format string `Hello %s!`, where the `%s` is a placeholder for a `String` input to the String format unit. We also insert an Output unit of type `Auto` and identifier `Greeting`.

After connecting the data flow units we get the data flow in figure 4.37. If this data flow is given the string `david` as input, it will output the string `Hello david!`.

Now we open the control flow in the main window, and we add a `Medarbejderportal Message` interaction by going to **Insert → Service interaction**. Then we connect the `Unary_input` interaction's `Success` branch to the `Greeting_Message` data flow, and we connect the `Greeting_Message` data flow to the `Message` interaction.

The `Greeting_Message` data flow expects a `Username` input, this input can be selected from the `Unary_input` interaction, since the `Unary_input` interaction outputs the user-name the Netbank user has entered.

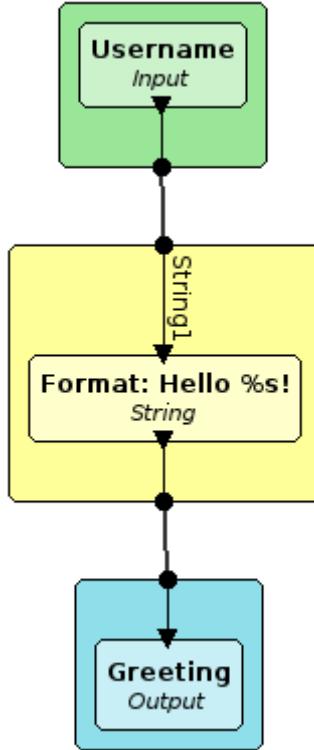


FIGURE 4.37: BIDE Example Data Flow 4 (Finished)

The **Message** interaction expects two inputs: a username and a message. The username can be selected from the **Unary_input** interaction, and the message can be selected from the **Greeting_Message** data flow.

Once everything is connected, the only things missing are to insert and connect Stop and Fail units. We can insert a Stop unit by going to **Insert → Stop**, and we can insert a Fail unit by going to **Insert → Fail**. Stop units do not have input, and Fail units expect an input of type **Error**, the **Error** branch always provides an output of type **Error**.

The **Error** type cannot be used to distinguish between different types of errors in BIDE, it can only be used to indicate that some error has happened with an error message. It is useful to be capable of distinguishing between different types of errors, hence we expect later versions of BIDE to be capable of distinguishing between different types of errors.

The finished control flow is shown in figure 4.38.

Implementation

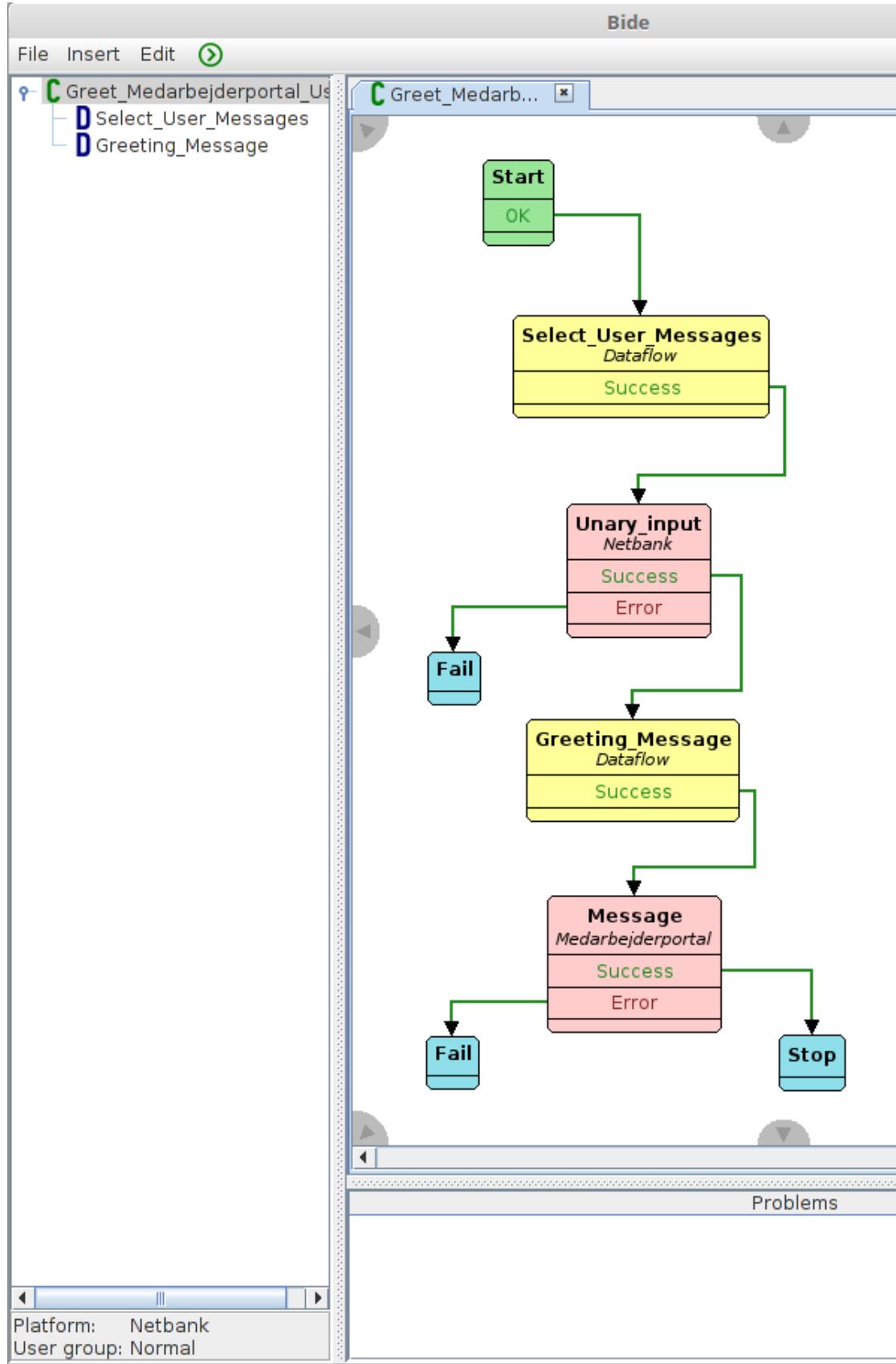


FIGURE 4.38: BIDE Example Control Flow 3 (Finished)

4.1.5.2 Application Test

To compile the application we implement in section 4.1.5.1 and make the application available on Netbank, we can press the green and round button which is located in the top menu of BIDE. We can see the compile button in figure 4.38.

If there are problems with the control flow or some data flow when compiling, then the problems will be listed below the main window, and the application cannot be compiled. To fix the problems, the user can click on a problem in the list which will take him to the location of the problem.

Figure 4.39 illustrates a problem where the user has forgotten to connect the **Error** branch from the **Message** interaction to the **Fail** unit. By clicking on the second problem the user is taken to the **Message** interaction, which is highlighted to indicate where the problem is located.

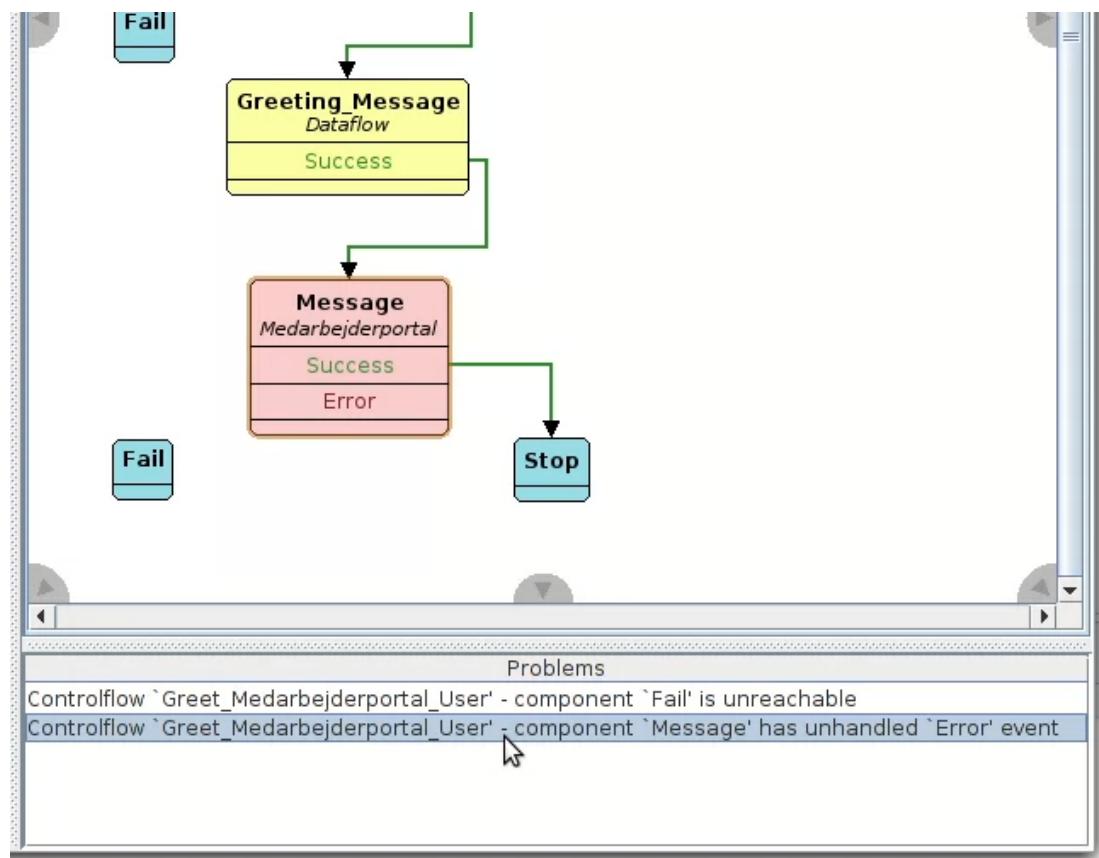


FIGURE 4.39: BIDE Example Compilation Problems

There are no problems with the implementation from section 4.1.5.1. Hence, when the compile button is pressed and the application has finished compiling, a link to start the application will be added to the imaginary Netbank home page.

When all the necessary servers are started, we can test the application by opening a browser and entering the URL: <http://localhost:5001/Netbank/yuezhu>. With this URL we log on to Netbank as user `yuezhu`.

The Netbank page looks as shown in figure 4.40, assuming that the only available application is the one we just compiled. If there are more Netbank applications, there will be links for starting them under the `new tasks` heading.

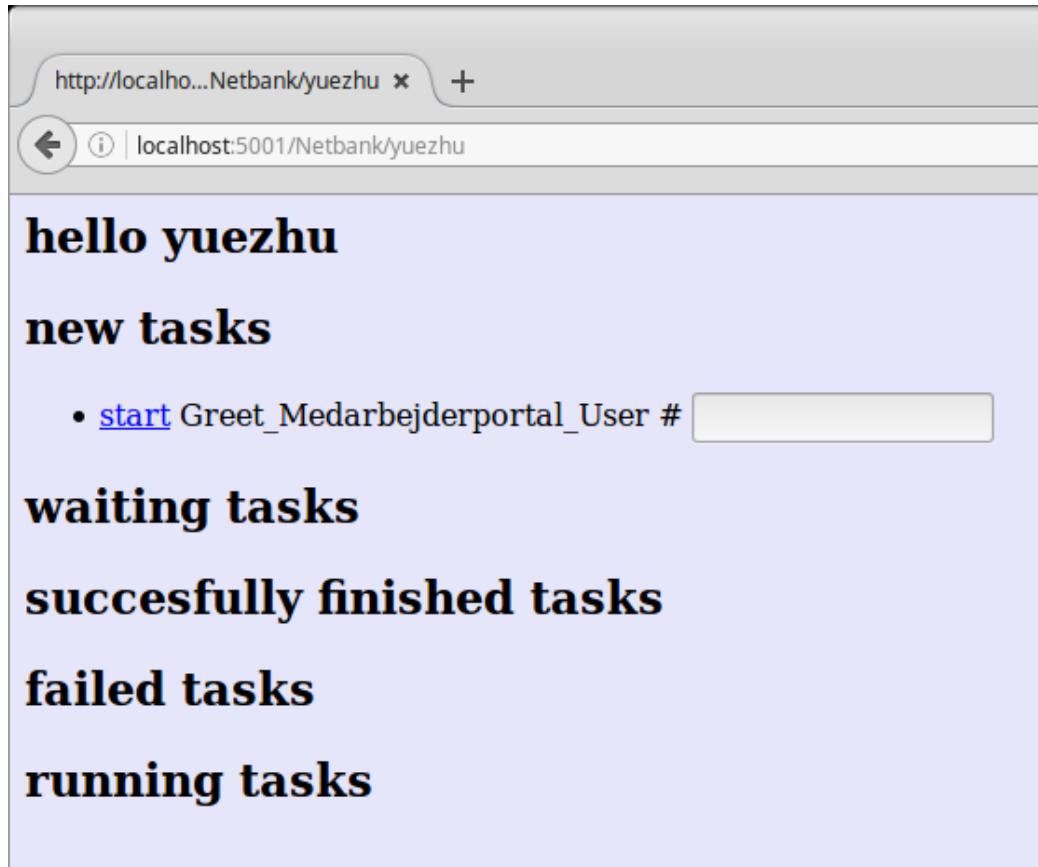


FIGURE 4.40: BIDE Example Netbank Home Page

To start an application the Netbank user must enter an application identifier in the text label after the `#`-sign and press on the `start` link. The identifier is used to allow a user to start multiple instances of an application and distinguish between the application instances using the identifiers. Some applications are better served with an automatically generated identifier, however as BIDE is a PoC, this suffices.

By entering identifier `greet` and pressing the `start` link the application is started. The first thing the application does is to ask the Netbank user to enter a username of a Medarbejderportal user. This is shown in figure 4.41.

By entering `andreas` and pressing `OK` the Netbank user is shown the page in figure 4.42.



Who would you like to greet on Medarbejderportal?

Enter the username here:



FIGURE 4.41: BIDE Example Enter Username Page



task Greet_Medarbejderportal_User # greet is running

FIGURE 4.42: BIDE Example Task Running Page

The Netbank user can wait on this page for a response when the Medarbejderportal user has seen the greeting, or he can press on the Home button. Since the application will terminate after the Medarbejderportal user has seen the greeting, it makes most sense to press the Home button, which will take the Netbank user to the Netbank home page, this is illustrated in figure 4.43. By pressing the link in the bottom of the screen the Netbank user can be taken back to the page in figure 4.42.

hello yuezhu

new tasks

- [start Greet_Medarbejderportal_User #](#)

waiting tasks

successfully finished tasks

failed tasks

running tasks

- [Greet_Medarbejderportal_User # greet](#)

FIGURE 4.43: BIDE Example Netbank Running Task

Now, if we enter URL `http://localhost:5001/Medarbejderportal/andreas` in the browser, we log on as user `andreas` on Medarbejderportal. This is shown in figure 4.44.

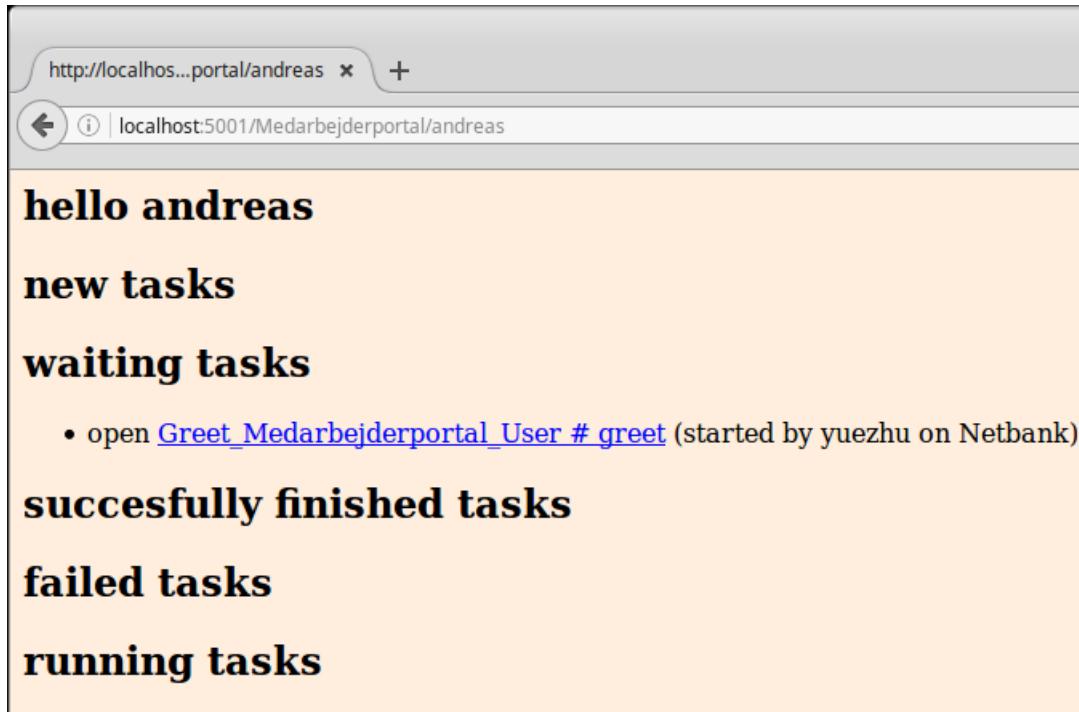


FIGURE 4.44: BIDE Example Medarbejderportal Waiting Task

By clicking on the link under the **waiting tasks** heading, the Medarbejderportal user is taken to the greeting page. The greeting page is shown in figure 4.45.

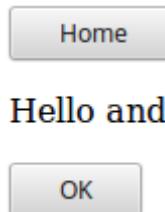


FIGURE 4.45: BIDE Example Greeting Page

When the Medarbejderportal user has finished studying the greeting, he can press the **OK** button to terminate the application.

When the application has terminated, the Netbank user can see it on the Netbank home page by looking under the **successfully finished tasks** heading.

While discussing entering a Medarbejderportal username in the page in figure 4.41, we assumed that the entered username existed. If the entered username does not exist,

then the page in figure 4.46 is shown to the Netbank user, and an error message will be added to the Netbank user's home page under the `failed tasks` heading. This is what we implemented in figure 4.38 where the `Error` branch from the `Message` interaction goes to the Fail unit. An `Error` branch can go anywhere that other branches can go.



FIGURE 4.46: BIDE Example Task Failed Page

Figure 4.47 shows the error message when `abc` is entered in the page in figure 4.41.

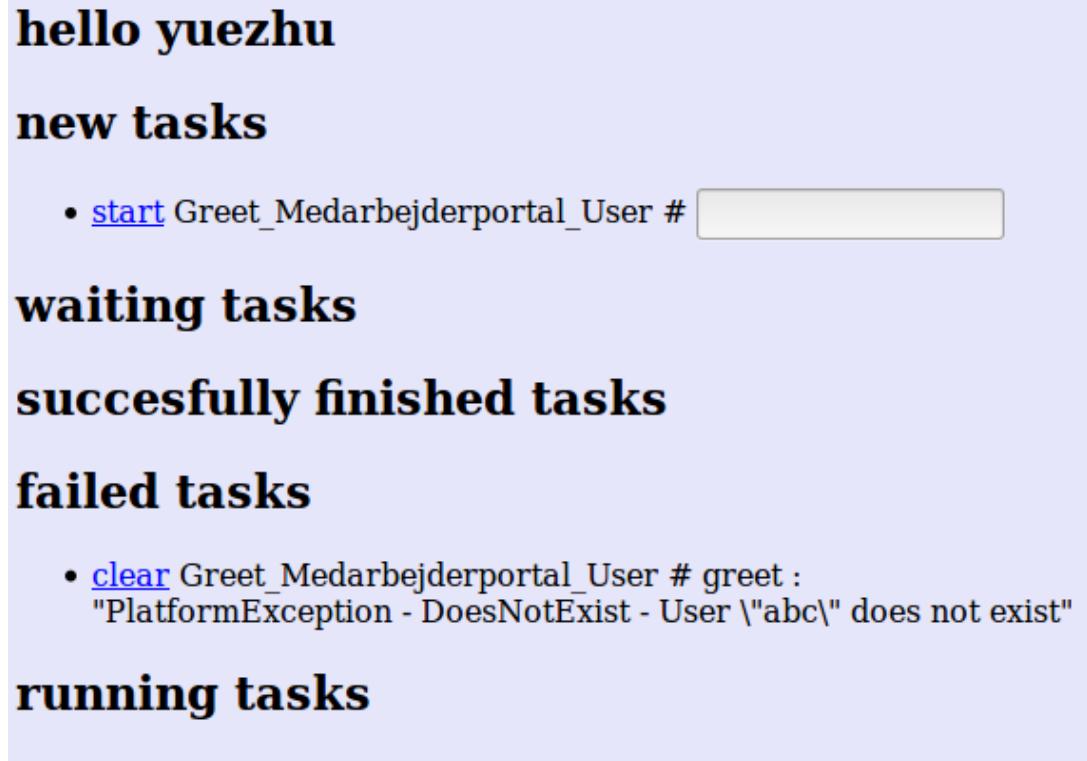


FIGURE 4.47: BIDE Example Netbank Failed Task

4.2 BIDE Internals

Since BIDE is a medium sized project requiring 2D graphics and GUI, BIDE is implemented using Java. Java is well suited for such projects, because Java has a reasonably safe static type system and Java's standard library comes with widely used 2D graphics and GUI APIs. Appendix E contains the Java code which implements BIDE.

Much of the BIDE source code consists of hand coded user interfaces. When implementing complex user interfaces, like data mapping in section 4.1.4, it is beneficial to hand code them rather than using a drag and drop GUI builder. By hand coding the GUI we get access to GUI functionalities which are not supported by GUI builders.

4.2.1 Abstract Syntax Tree

Before saving a project or compiling a project, BIDE generates an Abstract Syntax Tree (AST) which captures all the necessary information from the control flow graph and the data flow graphs.

When saving a project the AST is converted to XML format which is written to a file. When opening a project the XML file is parsed to obtain the information to exactly regenerate the control flow graph and the data flow graphs.

When compiling a project the AST is converted to Haskell code, which gets pre-processed and compiled into multiple executables. See section 4.2.2 for a description of the Haskell code representation.

4.2.2 Haskell Representation

4.2.2.1 Example of Haskell Representation

When compiling a BIDE application, the BIDE AST is converted into a special Haskell code representation. A hand coded Haskell implementation of an application with the exact same behaviour as the BIDE application implemented in section 4.1.5.1, is shown in listing 4.1.

```
1 module Greet where
2
3 import SeqServiceIO
4 import ServiceError
5 import Json
6
7 start :: IO (Either ServiceError ())
8 start = runSeqServiceIO $ do
9     let msg = "Who would you like to greet on Medarbejderportal?"
10    let lbl = "Enter the username here"
11    setVarSIO "msg" (strToJson msg)
12    setVarSIO "lbl" (strToJson lbl)
13 {-- Interact start :: Netbank.Unary_input
14 {User=_username, Message=msg, Label=lbl}
15 {Error=startError, Success=showGreeting} ->
16 {user=Input}
17 --}
```

```

18
19 startError :: IO (Either ServiceError ())
20 startError = return (Right ())
21 {-- Interact startError :: Runtime.fail
22   {Error=start_Error}
23   {Event=terminate} ->
24   {}
25 --}
26
27 showGreeting :: IO (Either ServiceError ())
28 showGreeting = runSeqServiceIO $ do
29   user <- getVarSIO "user" >>= eitherSIO . jsonToStr
30   let greeting = "Hello "++user++"!"
31   setVarSIO "greeting" (strToJson greeting)
32 {-- Interact showGreeting :: Medarbejderportal.Message
33   {User=user, Message=greeting}
34   {Error=greetingError, Success=terminate} ->
35   {}
36 --}
37
38 greetingError :: IO (Either ServiceError ())
39 greetingError = return (Right ())
40 {-- Interact greetingError :: Runtime.fail
41   {Error=showGreeting_Error}
42   {Event=terminate} ->
43   {}
44 --}

```

LISTING 4.1: Haskell Code Representation

The entry point of the application is the `start` function. The `start` function stores two strings with keys `msg` and `lbl` in a runtime database using the `setVarSIO` function. The `strToJson` function is used to convert a Haskell string to a Json value, since the values in the database are stored as Json.

The text between the `{--` and `--}` under the `start` function is not a comment, it is code specifying what to do when the `start` function returns. It specifies that the Netbank interaction `Unary_input` will be invoked by passing the database value of key `_username` as `User` argument, where the database key `_username` automatically contains the username of the Netbank user who started the application. The code also specifies that the two strings from the database stored under keys `msg` and `lbl` are passed as `Message` and `Label` arguments to the `Unary_input` interaction. If the interaction fails, then the `startError` function will be called. Otherwise, if the interaction succeeds, then the `showGreeting` function will be called, and the input the Netbank user has entered will be stored in the database under the key `user`, this is what the `{user=Input}` line specifies.

If the `startError` function is called, then the application will fail with the error message from the `Unary_input` interaction, since `fail` is invoked after `startError` returns, as indicated by the code under the `startError` function between the `{--` and `--}`. The `start_Error` database key is automatically created when the `Unary_input` interaction is invoked after the `start` function fails. The value of the `start_Error` is the error code from the interaction.

If the `showGreeting` function is called, the database value of key `user` is obtained with the `getVarSIO` function and used to create the greeting message which is stored under database key `greeting`. When the `showGreeting` function returns, the code under the function between the `{--` and `--}` shows that the `Medarbejderportal` interaction `Message` will be invoked. The arguments passed to the `Message` interaction specifies that the user to receive the message is the username stored under the database key `user`, and the message, which will be shown to the user, is the database value with key `greeting`. If the interaction fails, then the `greetingError` function will be called. If the interaction does not fail, the application will successfully terminate.

If the `greetingError` function is called, then the application will fail with the error message from the `Message` interaction.

All BIDE applications are converted to this special Haskell representation when compiled. The code between the `{--` and `--}` is an extension we have made to Haskell. To compile this Haskell code we use a Python tool we have implemented. The Python tool pre-processes the Haskell code to extract the code between the `{--` and `--}`, then it uses the extracted code to generate multiple Haskell files. These Haskell files are, in turn, compiled with the Glasgow Haskell Compiler [15] to generate multiple executables. The Python tool stores the path to the executables in the runtime database which is being accessed by a runtime server. The runtime server is responsible for executing generated applications, where each application consists of multiple executables generated by the Python tool. The Python tool is attached as appendix C.

4.2.2.2 Analysis of Haskell Representation

Using a general purpose programming language like Haskell as intermediate representation makes it simple to use this language for implementing complex operations which can be used in BIDE. As we can see from listing 4.1, the Haskell representation is compact, so it may be tempting for experienced programmers to directly implement applications or parts of applications using the Haskell representation.

Haskell is well suited as intermediate representation of BIDE applications since ACID transaction handling can be automatically handled using a *Monad*, and the immutability of Haskell variables is well suited for implementing BIDE data flows which cannot violate the immutability of data flows' inputs and outputs.

However, a new text language which can be used as intermediate representation instead of Haskell, may be an advantage. A new text language can be specialized for the domain of BIDE applications and, for example, offer static type checking of runtime database values.

4.2.3 Application Execution

In this section we will examine what happens while a BIDE application is executing.

Figure 4.48 shows what happens when a user logs on to a platform.

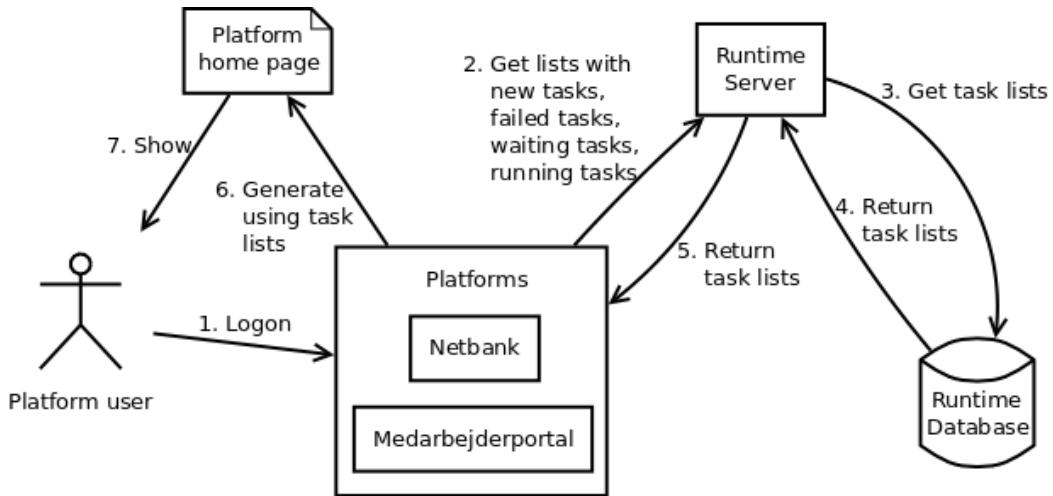


FIGURE 4.48: Platform Logon

1. The platform user logs on to a platform like Netbank or Medarbejderportal.
2. The platform asks the runtime server for lists of new tasks, failed tasks, waiting tasks and running tasks.
3. The server asks the runtime database for the task lists.
4. The database returns the task lists to the server.
5. The server gives the task lists to the platform.
6. The platform uses the task lists to generate the home page.
7. The home page is shown to the platform user.

From the home page, a platform user can see which new tasks he can start, which tasks have failed, which tasks are waiting for a response from him and which tasks are running. Figure 4.49 illustrates a typical scenario when a platform user starts a new task from the home page.

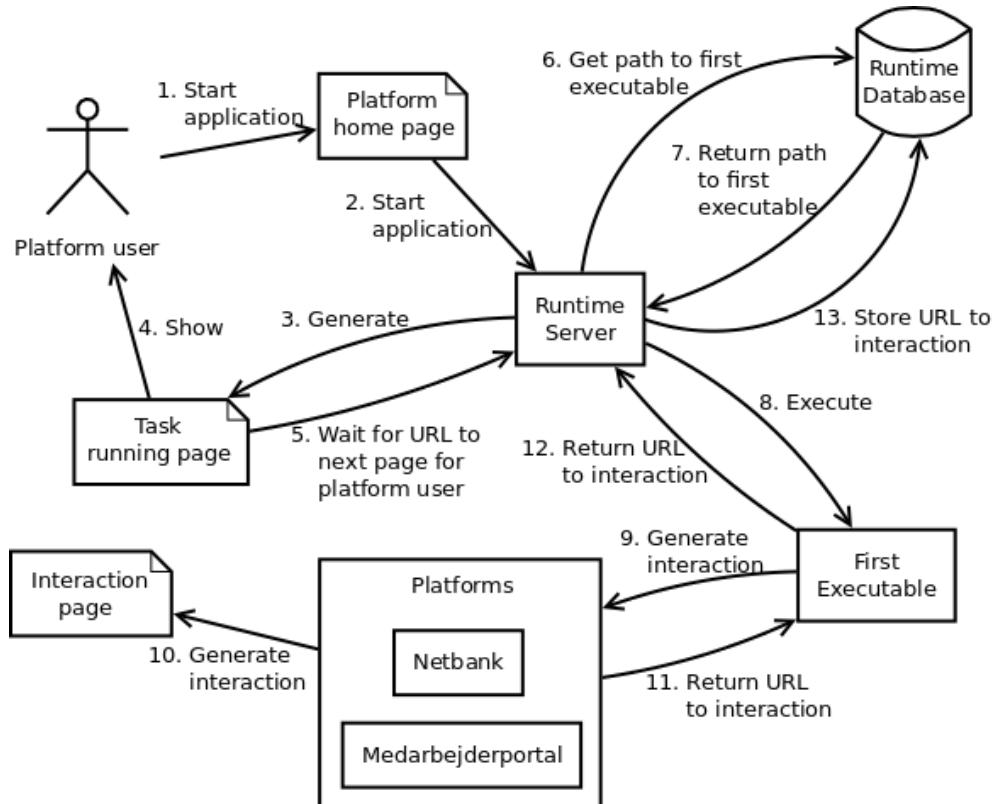


FIGURE 4.49: Start Application

1. The platform user starts the new application from the home page.
2. The home page tells the runtime server that the user wants to start the application.
3. As response, the server will generate a Task running page which says that the application is running.
4. The Task running page is shown to the platform user.
5. The Task running page is programmed to wait for the next page for the platform user from the application. If the platform user stays at this page, once a new page for that platform user from the application is ready, the new page will be shown to the platform user. This is useful for applications having multiple pages right after each other for the same user.
6. In order to start the application, the server asks the runtime database for the path to the first executable of the application.
7. The database returns the path to the first executable.
8. The server executes the executable.

9. The executable will ask a particular platform to generate a particular interaction.
10. The platform will generate the interaction.
11. And the platform will return the URL for opening the interaction to the executable.
12. The executable will return the URL to the server.
13. The server stores the URL in the database.

If the generated interaction page is for the same platform user, and the platform user has the Task running page open, then the platform user will be shown the interaction page. Otherwise, the interaction page is for another user, and that user can open the interaction page by logging on to the interaction's platform and opening the interaction page from the list of waiting tasks, from the platform's home page.

What happens after the interaction page is similar to starting a new application. When some button from the interaction is clicked, then the runtime server is told about it. The server will provide the user with the Task running page and ask the database for the next executable to execute, which can depend on which button was clicked. Then the executable can ask a platform to generate an interaction, and so on.

The executables do not always ask a platform to generate an interaction. An executable corresponds to one Haskell representation function and the following code between the {-- and --}, which is executed right after the function has returned. Refer to section [4.2.2.1](#) for an example of the Haskell representation. The code between the {-- and --} determines what the executable will do before terminating, it supports: generating an interaction, failing the application and more.

Appendix [D](#) contains Haskell code which implements the runtime server and the platforms.

4.3 Evaluation

In this section we will evaluate BIDE by demonstrating it to SDC and validating whether it satisfies the implementation requirements from section [2.2.4](#).

4.3.1 Evaluation by SDC

We have demonstrated BIDE at a meeting with representants from SDC's management, technical solution architects and our supervisors at SDC. They realize the similarities

with BPMN and BPEL and believe that IT developers at the banks would like to use BIDE.

We also have demonstrated BIDE to a sales manager at SDC. He likes the idea of banks implementing their own workflow applications. He thinks that BIDE is a good tool for SDC and bank employees with an IT background, but he says that BIDE is too technical for ordinary bankers.

The sales manager suggests that BIDE can be used to implement a workflow application to obtain (on-board) new bank customers: A potential customer can scan his passport and electronically send it to the bank. When the bank receives the scanned passport, the passport can be validated by a bank employee. The bank employee can accept or reject the potential customer. If the potential customer is accepted, then he will get added as customer in the bank system.

The sales manager have also mentioned that BIDE can support generating applications which register the bankers' progress. For example, BIDE applications which bankers use to add new customers can register how many new customers each banker on-boards. This data can be made available to the bank management to supervise their employees.

From our meetings at SDC we have got positive feedback in form of familiarity feelings with BIDE and recognition of BIDE's usefulness, and we have got new ideas for user stories. But, the reactions from our meetings disprove our hypothesis in section 3.3.1, which presumes that the design is suited for bankers. We will further address this issue in section 5.1.

4.3.2 Evaluation with Requirements

Here we will evaluate BIDE based on the implementation requirements from section 2.2.4.

Each row of table 4.1 contains a list of requirements in column **Requirements**. Since BIDE is a PoC, all of the required features are not implemented. For the implementation requirements, which require missing features, the **Fulfillment description** column is used to suggest how it can be implemented. For the other requirements, the **Fulfillment description** column will be used to describe how the requirements are fulfilled.

Requirements	Fulfillment description
IR1	BIDE allows developing applications which are started from different platforms by allowing the developer to select which platform an application will be started from.

IR2	By selecting user group Normal when developing an application with BIDE, then the application can be started by all users of the starting platform.
IR3	By selecting user group Administrator when developing an application with BIDE, then the application can only be started by administrator users of the starting platform. Adding more user groups and allowing selection of multiple of user groups is possible.
IR4	By requiring a password and username to log in and use BIDE we can restrict access of BIDE to selected employees.
IR5	Both data flows and control flows can be used to implement reusable operations. One way to make use of such reusable operations would be to add a feature to import them into projects.
IR6, IR8	BIDE can have a feature for allowing banks to share applications, and another feature for allowing banks to import applications shared with them. With these features, banks can share applications, and SDC can implement applications for banks.
IR7	It is possible to add a feature which puts BIDE applications in a test environment for validation before they are put in the production environment. Such a feature can make use of one of SDC's existing test environments.
IR9, IR10	A wizard for adding web services to BIDE can be implemented. This can be used by SDC to add services to banks, or it can be used by banks to add services themselves.
IR11	By requiring service documentation when adding a service to BIDE, then the documentation can be made available to developers using BIDE.
IR12	Confirming to security policies will require generated applications to implicitly pass security certificates to services.

TABLE 4.1: Implementation Requirement Fulfillment Descriptions

Chapter 5

Discussion

5.1 Comparisons with Related Languages

From the evaluations of VOL and BIDE in sections 3.2 and 4.3 we know that VOL has similarities and differences with the languages from section 2.3, and we expect that an implementation of VOL will be capable of fulfilling a high majority of the user requirements from section 2.2.

In this section we will identify the features that make VOL unique. We will find advantages of VOL, compared with the languages from section 2.3.

We know from section 4.3 that VOL does not fulfill requirement DR1, which requires VOL to be easy for bankers. The other languages from section 2.3 are more complex than VOL, hence those languages also cannot fulfill requirement DR1. Since developers at SDC and developers at the banks can use VOL, we do not see it as a major problem that bankers cannot.

5.1.1 Unique Features of VOL

A unique feature of VOL is that VOL has service invocations in the data flows rather than in the control flows. In VOL, all data flows are ACID transactions.

Another unique thing about VOL is its simplicity. This is partly because the design of VOL leaves details as implementation defined, but also because we have intentionally aimed for a minimalistic design of VOL.

5.1.2 Analysis

In the following sections we will analyse the languages from section 2.3. We will get to understand how they support ACID transactions of service invocations, error handling and multi platform workflows. These features are all important to a language for the domain.

Table 5.1 summarizes how VOL supports the features.

Feature	Solution
ACID Transactions	Service invocations are in data flows, where data flows are always ACID transactions.
Error Handling	Error handling is done in the control flow with the error branch. The developers are forced to think about which actions to take on errors.
Workflows	The control flow models a multi platform workflow, where the developer specifies the order of interactions.

TABLE 5.1: Discussion Analysis of VOL

5.1.2.1 Analysis of SSIS

Table 5.2 contains the analysis of SSIS.

Feature	Solution
ACID Transactions	Service invocations are located in the control flow, but SSIS does not support ACID transactions in the control flow.
Error Handling	Error handling is done in the control flow with an error branch. Unlike VOL, the developers are allowed to ignore errors.
Workflows	SSIS does not support workflows.

TABLE 5.2: Discussion Analysis of SSIS

Since SSIS does not support ACID transactions of service invocations or workflows, SSIS is not a solution for the domain defined by the user requirements.

5.1.2.2 Analysis of LabVIEW

Table 5.3 contains the analysis of LabVIEW.

Feature	Solution
ACID Transactions	LabVIEW does not have transaction support for service invocations.
Error Handling	Error handling is commonly done by checking a returned error code.
Workflows	LabVIEW does not have workflow support.

TABLE 5.3: Discussion Analysis of LabVIEW

Since LabVIEW does not support workflows it cannot be used for the domain.

5.1.2.3 Analysis of BPEL

The default ACID transaction handling in BPEL is Compensation Handlers. Compensation Handlers offer a type of transaction handling, where compensating services are manually invoked to rollback transactions. This works, but it puts a burden on the developer to carefully invoke compensating services on errors.

The automatic rollback of transactions in VOL makes it easy for developers, by removing the burden of manually invoking compensating services. This is an advantage of VOL compared to BPEL. In section 5.2.1.3 we will see how automatic transaction rollback can be implemented in VOL.

Oracle BPEL Process Manager extends BPEL with automatic transaction handling, which implicitly splits the business processes into multiple transactions, based on which operations are used in the business processes.

Table 5.4 analyses Oracle BPEL Process Manager.

Feature	Solution
ACID Transactions	Implicit transaction handling mechanism, based on which operations business processes use.
Error Handling	Error handling is supported with Fault Handlers for catching errors.
Workflows	Oracle BPEL Process Manager supports workflows, but only for their own platform. It does not support other platforms like Medarbejderportal or Netbank, etc.

TABLE 5.4: Discussion Analysis of Oracle BPEL Process Manager

All the implementations of BPEL, that we know of, only support workflows for a pre-defined platform. SDC wants support for workflows across multiple of the existing platforms. Hence, BPEL is not a solution for the domain.

5.1.2.4 Analysis of BPMN

BPMN is a notation standard and not an executable language. However, there are executable implementations of BPMN, thus we will consider BPMN as a possible solution for the domain defined by the user requirements.

Camunda is an interesting implementation of BPMN [16]. Table 5.5 analyses BPMN.

Feature	Solution
ACID Transactions	BPMN supports ACID transaction handling by using Transaction Sub-processes.
Error Handling	BPMN supports error handling by using the Error End Event.
Workflows	BPMN Supports workflows, but most BPMN implementations does not support multi platform workflows. The Camunda BPMN implementation has a REST API which can be used to access the Camunda Core Engine [17]. Camunda might have support for developing multi platform workflow applications by making use of the REST API.

TABLE 5.5: Discussion Analysis of Camunda BPMN

To decide whether Camunda has the necessary multi platform workflow support, we will need to implement a PoC with Camunda. If it is possible, then Camunda is a solution for the domain. This is interesting because it is likely to be faster to implement a solution using Camunda rather than implementing a new DSL.

5.1.3 Evaluation

In section 5.1.2 we find that SSIS, LabVIEW and the BPEL implementations are incapable of fulfilling important requirements, which VOL fulfills. However, the Camunda implementation of BPMN might be able to fulfill the same requirements as VOL.

For the Camunda BPMN implementation to fulfill the same user requirements as VOL, it will require implementation of plugins to Camunda. Implementing the necessary

Camunda plugins will be faster than implementing a new DSL, but it will require sacrificing flexibility. By implementing a new DSL, we have all the power to modify it to our needs. By using Camunda, it is unavoidable that we will have to accept limitations of Camunda.

BPMN is a generic language for solving many tasks that are not required by the requirements. This introduces unnecessary complexity, although it makes BPMN more likely to support new requirements. We can extend VOL by adding support for fulfilling new requirements. If it becomes necessary to extend BPMN in Camunda, then it is not as easy.

Hence, a feature of VOL that Camunda BPMN does not have is: VOL can be freely implemented to fulfil current requirements and new requirements without limitations of a particular underlying framework.

After all, VOL and BPMN are similar languages, and they are both adequate DSLs for the domain defined by the requirements in section 2.2. If we find useful or better features in BPMN, then we can easily extend or change VOL to support them.

5.2 Future Work

In this section we will discuss important future work for further development of VOL.

Some work is important to get tested before implementing a first public release of VOL, this is what, BIDE, the PoC of VOL is for. BIDE is used to evaluate VOL, and test hypotheses.

In section 4.3.1 we use BIDE to evaluate VOL and disprove the hypothesis that VOL is easy for bankers.

Once BIDE has been tested and we are satisfied with the evaluations, then it is time to implement the first public release of VOL.

We will distinguish between work that needs to be done on the PoC, and work that needs to be done for the first release, rather than the PoC.

5.2.1 Proof of Concept

Here we will discuss features that are important to add to the PoC implementation, BIDE, such that we can evaluate and improve the design of VOL.

5.2.1.1 Real Services

The two main features of BIDE are workflows and service invocations. We need to have support for service invocations of SDC services in the PoC so we can evaluate it.

5.2.1.2 Work on Type System

We know that VOL's type system will not be sufficiently expressive. For example, VOL does not support recursively defined types. Since WSDL supports recursively defined types, to support invoking WSDL web services, VOL should support this.

Since VOL needs to support all WSDL types, WSDL is a good source of inspiration when developing VOL's type system. BPEL already supports the WSDL type system, thus BPEL's type system can also be used for inspiration.

5.2.1.3 ACID Transactions

Automatic rollback of ACID transactions is an important aspect of VOL. We want BIDE to support this such that we can test it.

Most of SDC's services do not support being part of ACID transactions, and we cannot expect external services to support this either. Unfortunately, this makes it impossible to support ACID transactions of multiple updating service invocations on VOL.

A solution to this problem is to rollback transactions by invoking compensating actions. This will require BIDE to know how to invoke compensating actions. For example, when adding a service which transfers X money from person A to person B , it is necessary to add a compensating action which transfers X money from person B back to person A . Another example, when adding a service which gives loan X to person A , the compensating action is to invoke a service which removes loan X from person A .

This solution suffers from the lost update and dirty read problems. Without services supporting ACID transactions, we cannot know when two transactions are concurrently invoking the same updating service, such that the update from one of the transactions is lost. Furthermore, we cannot avoid one transaction reading a value from another transaction which is later rolled back.

To reduce occurrences of the lost update and dirty read problems we can serialize all transactions, such that each bank can have 1 and only 1 transaction executing at a time. This does not guarantee that we can completely avoid the problems. The currently existing SDC applications do not support this kind of transactions, so we cannot know

when these applications are involved in a lost update problem or a dirty read problem. There are performance issues involved with serializing transactions, hence we do not think of it as a realistic solution.

Another solution is to implement new SDC services which supports this kind of ACID transactions, although this does not completely solve the problem, as we cannot expect external services to support ACID transactions. Either way, when a bank needs a new SDC service in the development environment, then SDC can implement it with ACID transaction support, and add it to all the banks' development environments.

5.2.1.4 Test Hypothesis

Section 3.3.2 establishes a hypothesis which, roughly, says that having service invocations in ACID data flows is better than having service invocations in the control flow with manual ACID transaction handling.

To test the hypothesis, we need two versions of BIDE. One version with service invocations in ACID data flows, and one version with service invocations in control flows with manual ACID transaction handling.

By having one set of developers using one version of BIDE and another set of developers using the other version, we can give them a series of test applications to implement. With test programs for testing the applications, we can get an estimate of which BIDE version has the fewest data inconsistency problems, related to erroneous transaction handling. In this way, we can decide which BIDE version to continue with.

5.2.2 First Release

Here we will look at features that are not important to test in a PoC of VOL, but are important features for the first public release of VOL.

5.2.2.1 Real Platforms

The first release of VOL needs to support user interactions in real platforms. This is not important in the PoC, since we can test BIDE using the current simulations of Medarbejderportal and Netbank.

5.2.2.2 Drag and Drop User Interactions

Having a number of predefined interactions which are supported for each platform is fine for a PoC, but it is hard to know exactly which interactions are required.

A first release of VOL should support implementing user interfaces by dragging and dropping GUI components. This makes VOL more generic and easier to use, we expect dragging and dropping components will be more intuitive than selecting an interaction from a long list of possibilities.

5.2.2.3 Implementation Requirements

As described in section [2.2.4](#), there are a number of implementation requirements which need to be implemented in the first release of VOL.

Chapter 6

Conclusions

6.1 Problem

As part of SDC's banking service development and maintenance, SDC is looking for two new DSLs:

1. A new DSL for replacing the Rules language, which they are using for implementing the core of services.
2. SDC also wants a new visual DSL for banks to access SDC services and external services.

Our initial task was to decide which of the two DSLs is the most important to design for SDC. Once decided, the objective was to design the most important of the two DSLs.

6.2 Procedure

To decide which of the two DSLs was the most important to design, and to get a better understanding of SDC, we analysed SDC's development procedure and internal system in section 1.3. In cooperation with SDC in section 2.1, we decided that a new DSL for banks is more important than a new DSL for replacing Rules.

Now that we had chosen which DSL to design, we found user requirements for the DSL for banks in section 2.2, and we researched related languages in section 2.3.

With user requirements and inspiration from related languages, we designed VOL, a new DSL for banks, in section 3.1. To make sure we were going in a satisfying direction

with the design, we evaluated the design by using user requirements and by critically comparing it with related languages in section 3.2.

Once the design of VOL was ready, we implemented BIDE, a PoC implementation of VOL, described in section 4.1. We used BIDE to further evaluate VOL by testing and discussing it with SDC in section 4.3.

In section 5.1 we discussed VOL by comparing it to related languages and found advantages of VOL. In section 5.2 we discussed future work important for improving the design of VOL and for implementing the first public release of VOL.

6.3 Findings

By comparing VOL with related languages in section 3.2.1 we can conclude the following:

- VOL's control flows offer a subset of the features offered by BPMN's process diagrams.
- While VOL has service invocations in the data flows, BPEL has service invocations in the control flow.
- VOL's data flows are conceptually similar to data flows in LabVIEW.
- The overall design of VOL with control flows and data flows reminds of the overall design of SSIS.

Service invocations in the data flow is the most controversial feature of VOL, and it has given rise of the hypothesis from section 3.3.2. The hypothesis says: When having service invocations in data flows and requiring all data flows to be ACID transactions, we decrease the number of problems with data inconsistencies, compared with having service invocations in the control flows and manual ACID transaction handling. This hypothesis still needs to be tested, as described in section 5.2.1.4.

In section 4.3.1 we demonstrated BIDE to SDC. We got mostly positive feedback. BIDE is useful for developers at SDC and developers at the banks, but the feedback disproved the hypothesis from section 3.3.1, which says that VOL is easy for bankers.

Section 5.1 analyses the languages from section 2.3 as solutions for the domain defined by the user requirements, and compares them with VOL. We can conclude that SSIS, LabVIEW and BPEL are inadequate for the domain while the Camunda BPMN implementation might be adequate. To determine whether Camunda is adequate as a solution for the domain, we will need to implement a new PoC using Camunda.

Using Camunda has the advantage that it already implements BPMN. By using Camunda it will only require implementing plugins for Camunda, which is faster than designing and implementing a new language like VOL.

An advantage of using VOL, compared to Camunda, is that it can be designed to fulfill the user requirements without limitations of a particular framework like Camunda. BPMN aims to model intercommunicating concurrent workflows, while VOL only aims to model sequential workflows. This makes VOL simpler than BPMN.

Appendix A

Grammar Specification

Contents

1	Introduction	91
2	Grammar Rule Definition	91
3	Control Flow Syntax	92
3.1	Control Flow Units	92
3.2	Control Flow Grammar Rule	93
3.3	Control Flow Examples	96
4	Data Flow Syntax	97
4.1	Data Flow Units	97
4.2	Data Flow Grammar Rule	98
4.3	Data Flow Examples	102
5	Interaction Flow Syntax	105
5.1	Interaction Flow Units	105
5.2	Interaction Flow Grammar Rule	105
5.3	Interaction Flow Examples	106
6	Type Syntax	108
6.1	Type Units	108
6.2	Type Grammar Rule	108
6.3	Type Examples	109
7	Constant Syntax	110
7.1	Constant Units	110
7.2	Constant Grammar Rule	111
7.3	Constant Examples	112
8	Identifier Syntax	113
8.1	Identifier Grammar Rule	113
8.2	Identifier Examples	113
9	Program Grammar Rule	115

1 Introduction

This specification defines the syntax of the Visual Orchestration Language (VOL).

The document does not assume a particular encoding of VOL source code. It describes the syntax, but the encoding of VOL source code is left implementation defined.

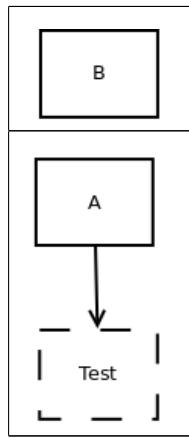
2 Grammar Rule Definition

Since this VOL grammar specification does not require a particular encoding of VOL source code, the VOL syntax is abstractly defined in terms of grammar rules which can expand to one of possibly multiple graphs.

When a grammar rule is defined it is given a name and a table of possible expansions. For example, we can define a grammar rule called `test`:

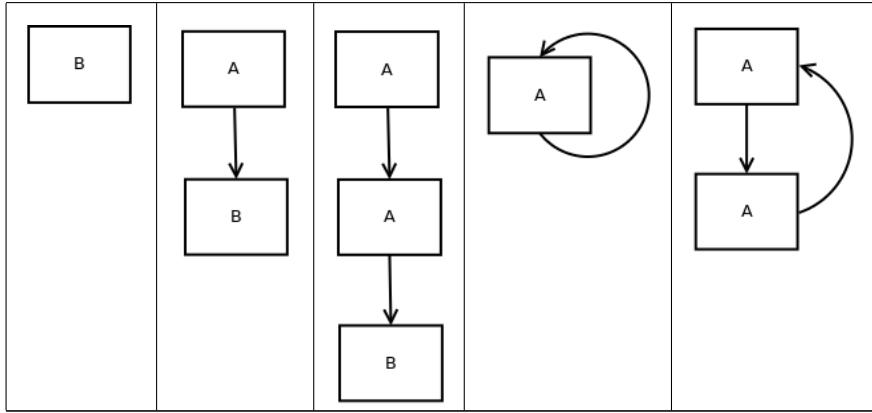


where `test` has the expansions in the following table:



Whenever `test` appears, it must be replaced by one of the expansions in order to form valid syntax.

When recursively applying the `test` rule, five of the possible expansions are:



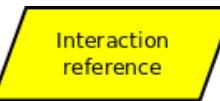
The following sections will define grammar rules which are used to define the syntax of VOL source code. In section 9 we will see a single grammar rule which defines the syntax of VOL programs.

3 Control Flow Syntax

3.1 Control Flow Units

This section describes the control flow units. The control flow units are listed in the following table including a short description of each unit's semantics.

Token	Description
	Entry of control flow.
	End of control flow.
	Raise an error.
	Execute data flow
	Execute sub control flow.

	Control flow branch based on condition.
	Perform environment interaction.

3.2 Control Flow Grammar Rule

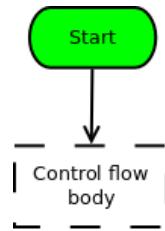
This section describes the `control flow rule` which is a syntax rule used to describe all syntactically valid control flows.

In this section, control flow units are modeled as described in 3.1.

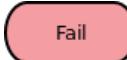
The `control flow grammar rule` which defines the syntax of all control flows is:

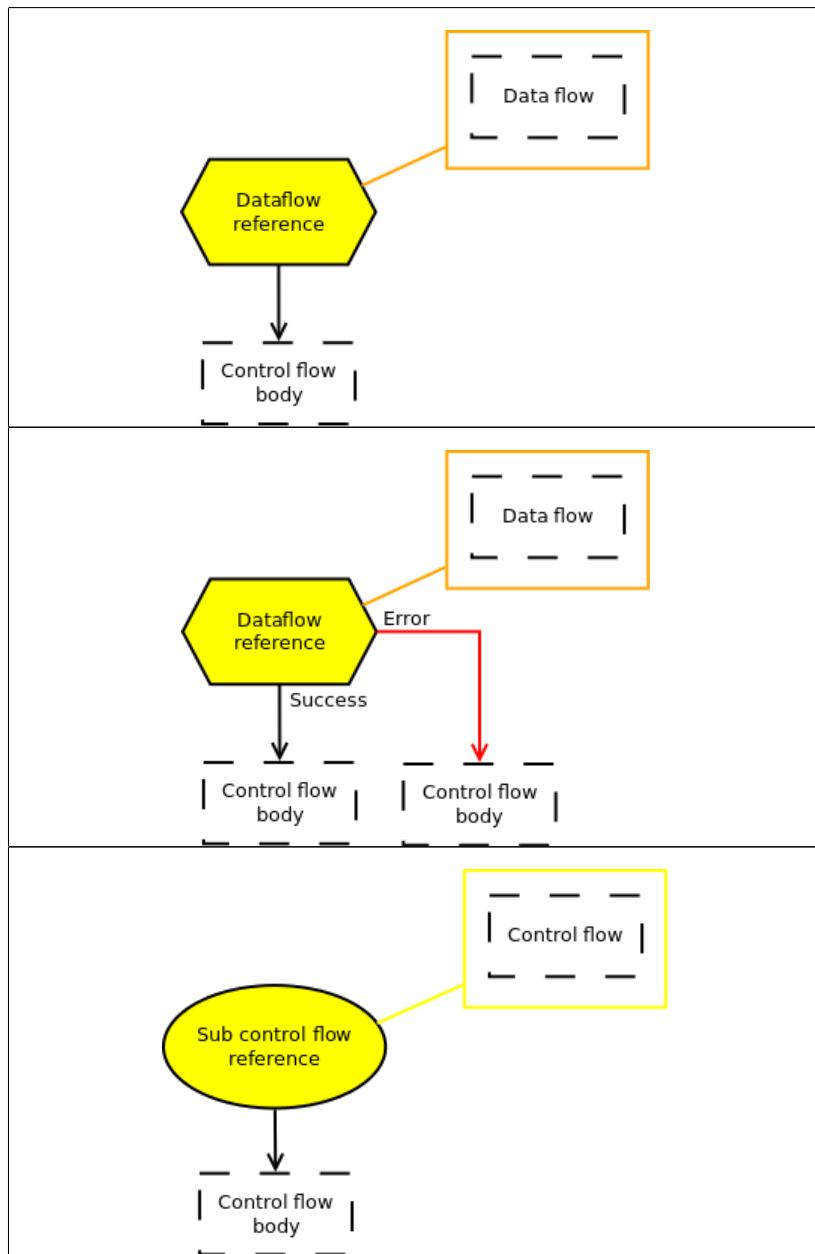


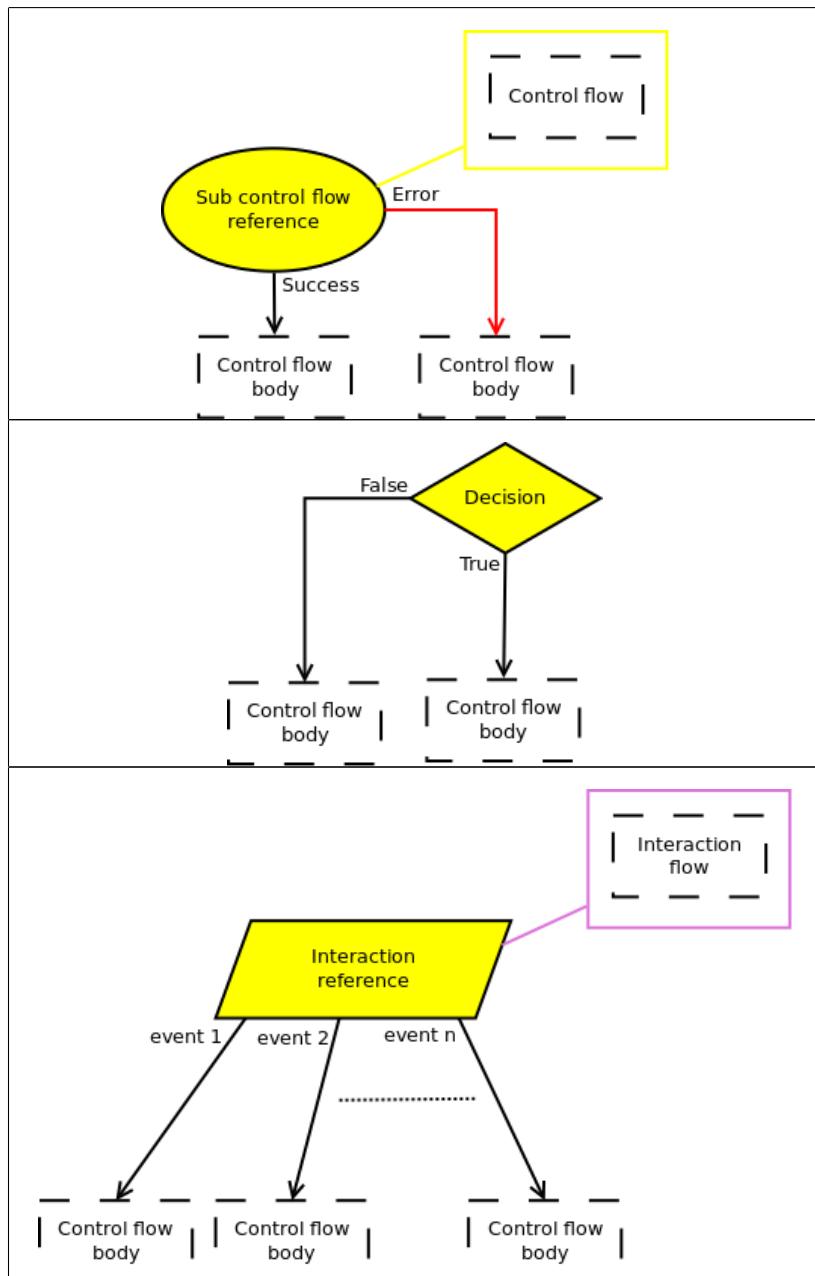
The `control flow rule` says that all control flows begin with the `start` unit followed by the `control flow body` grammar rule. The `control flow rule` has a single expansion which is:

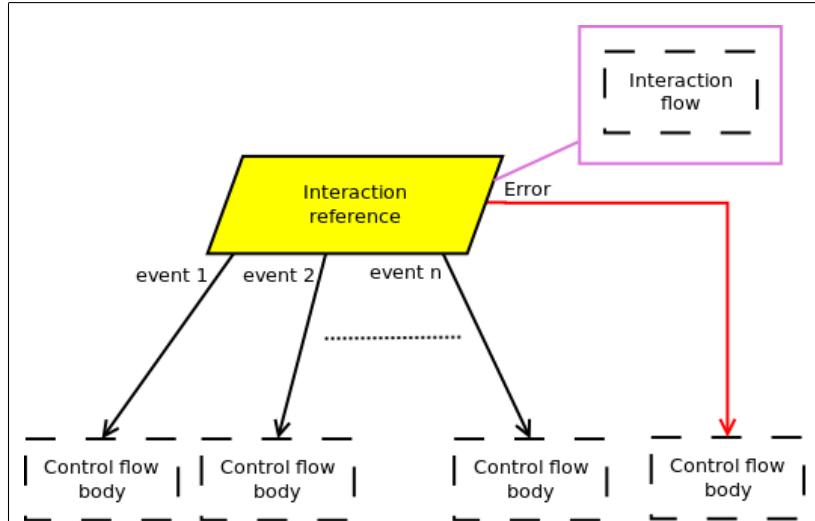


Where the `control flow body` grammar rule is defined to have the (possibly recursive) expansions in the following table:







Notice that the number of edges going out of the `interaction reference` unit is statically defined by the `interaction flow` rule.

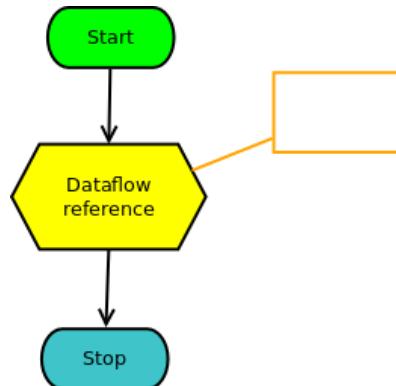
The `data flow` grammar rule is described in section 4.2 and the `interaction flow` grammar rule is described in section 5.2.

3.3 Control Flow Examples

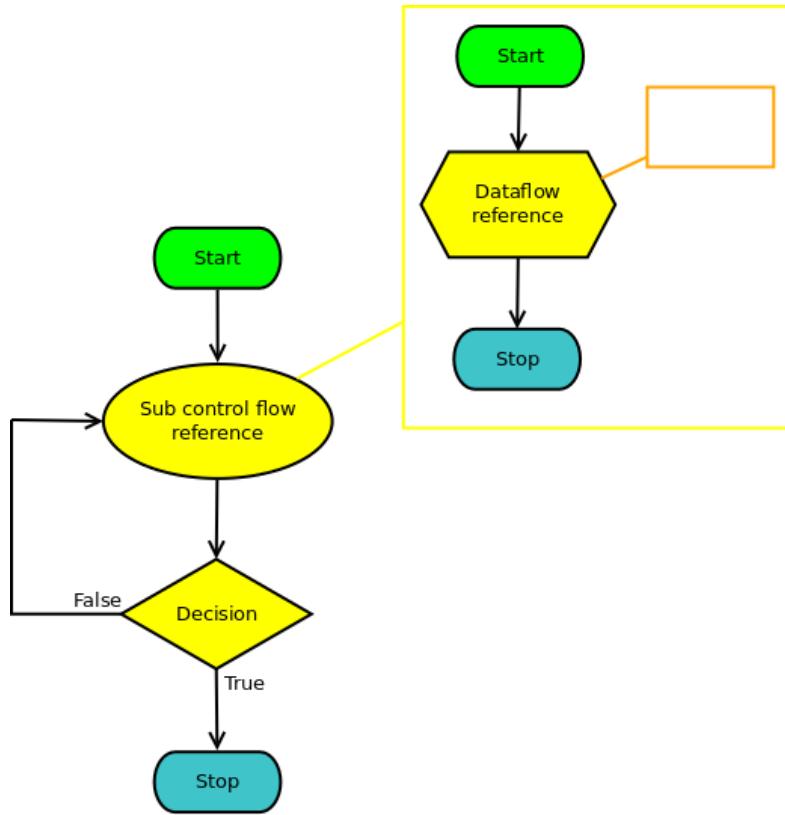
In the control flow examples in this section the `data flows` are left empty. Refer to sections 4.3 for examples of data flows which are not empty, refer to section 4 for a description of the `data flow` grammar rule syntax.

Following the grammar rules from section 3.2 we can construct syntactically correct control flows.

All control flows begin with the `start` unit followed by a `control flow body`, for example:



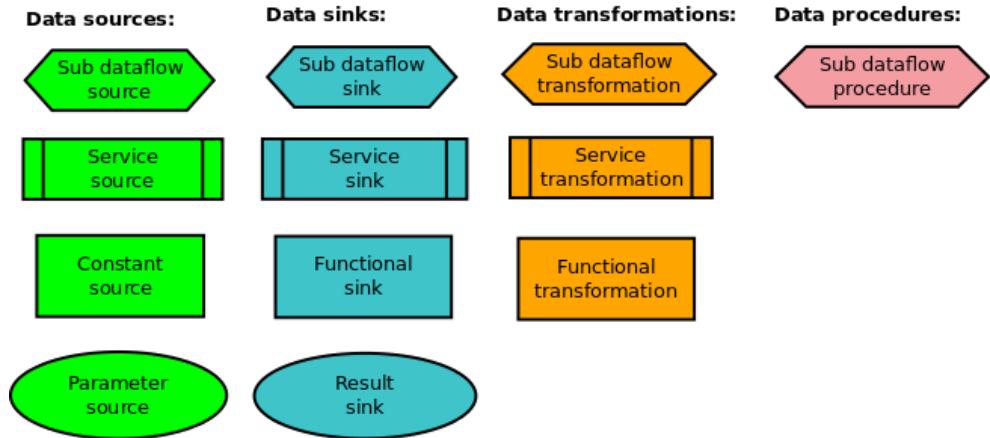
An example with a loop and a sub control flow reference is:



4 Data Flow Syntax

4.1 Data Flow Units

The following figure illustrates most of the data flow units, they are classified as source, sink or transformation. Sources are data flow units providing output, sinks receive input and data transformations receive input and provide output:



The last data flow unit is the **port** unit:



The **port** units are used as intermediate identity vertices. The **port** units each has one input and one output.

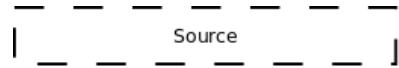
4.2 Data Flow Grammar Rule

This section describes the **data flow** grammar rule which defines the syntax for modelling data flows:

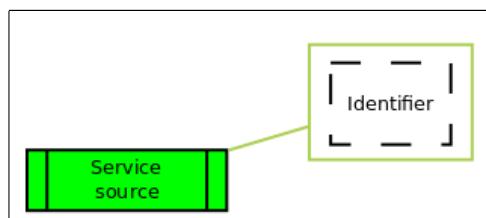


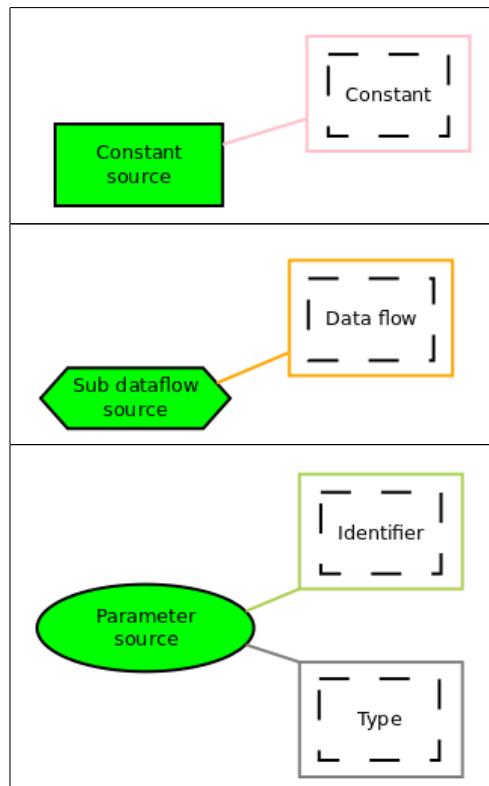
In order to define the **data flow** rule we will use the **type**, **constant** and **identifier** grammar rules which are defined in sections 6.2, 7.2 and 8.1 respectively.

Before defining the **data flow** rule we will define the **source** rule:



where the **source** rule has the expansions in the table:

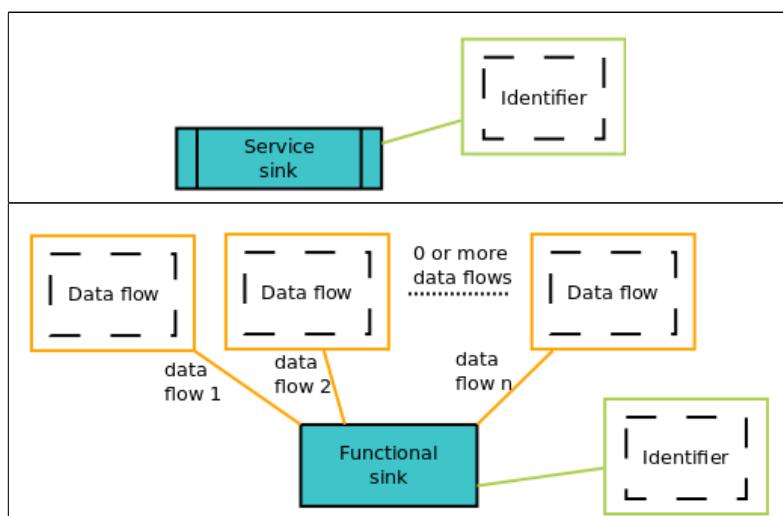


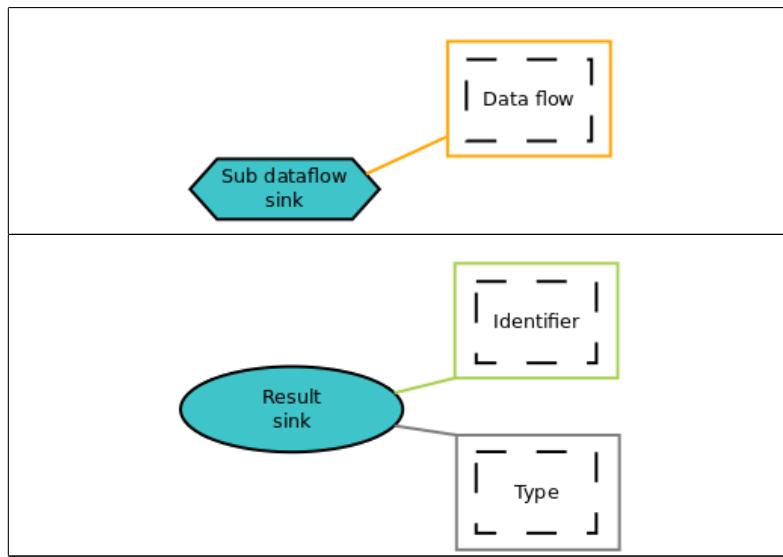


We will further define the **sink** rule:



to have the expansions:

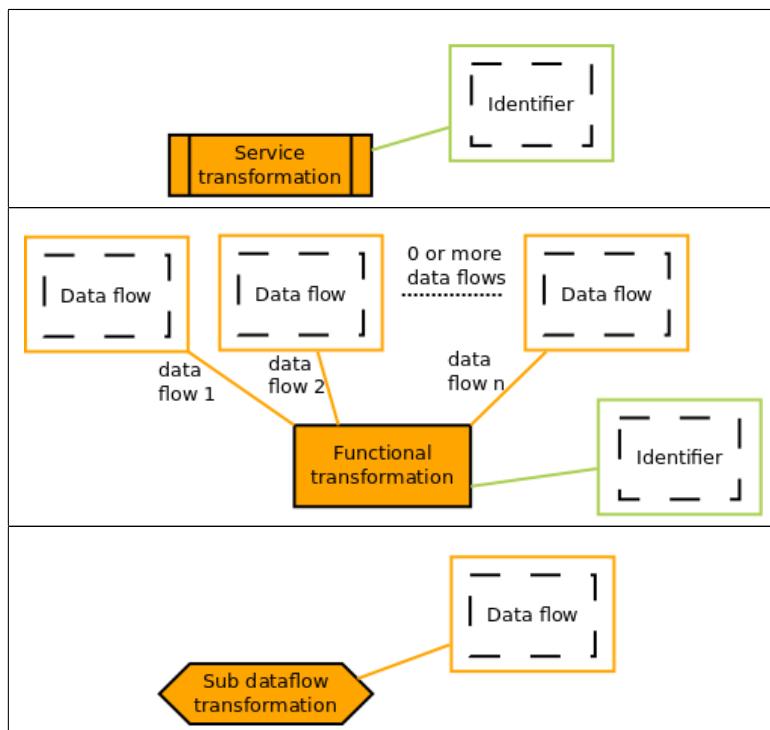




We will define the **transformation rule**:



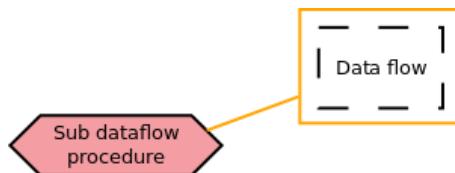
with the expansions:



And we will define the **procedure** rule:



to have a single expansion:

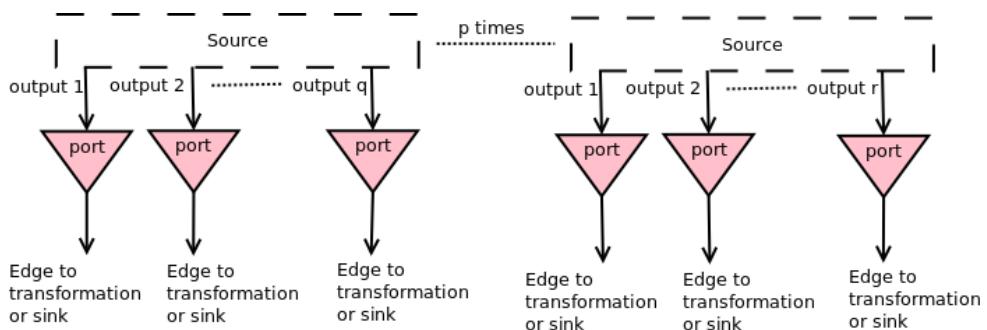


Now we will define the **data flow** grammar rule. The **data flow** grammar rule has a single expansion.

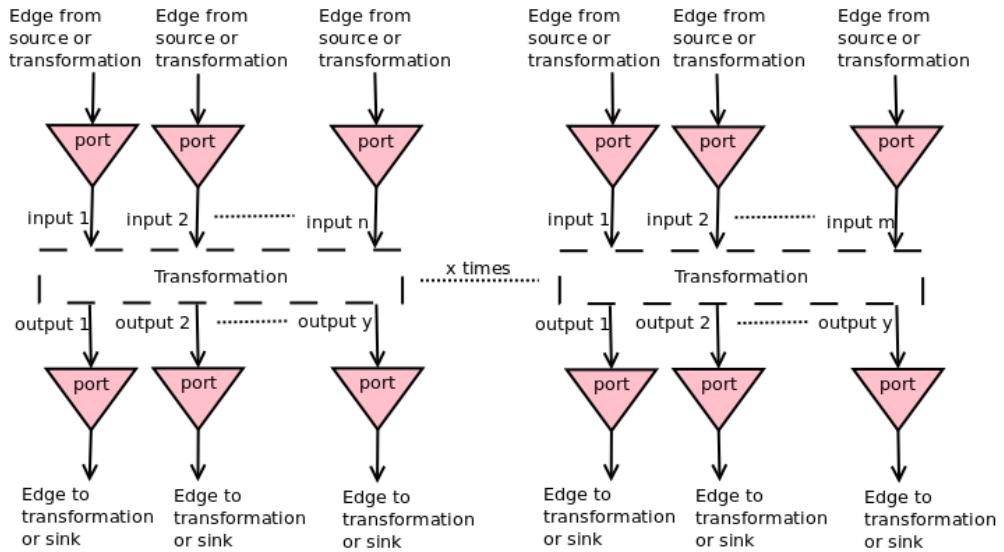
The **data flow** grammar rule expansion contains any number of **procedures**:



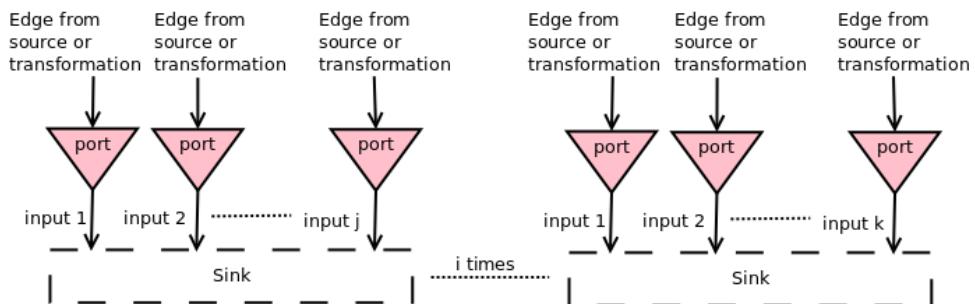
The **data flow** grammar rule expansion contains any number of **sources**, where each source is connected to at least 1 port unit with a directed edge:



The **data flow** grammar rule expansion contains any number of **transformations**, where at least 1 port is connected to each **transformation** with a directed edge and each **transformation** is connected to at least 1 port with a directed edge:



And the **data flow grammar rule expansion** contains any number of **sinks**, where at least 1 **port** is connected to each **sink** with a directed edge:



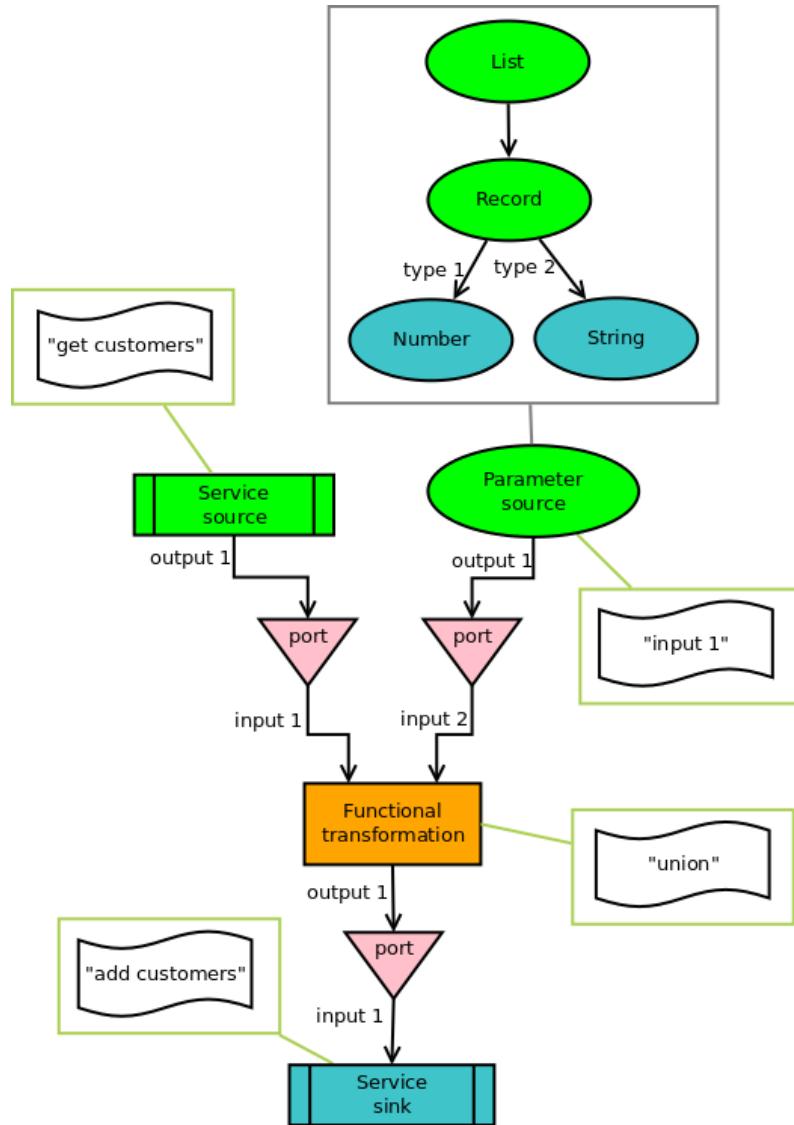
All **port** units must have exactly 1 incoming edge and 1 outgoing edge. In section 4.3 we will see data flow syntax examples.

4.3 Data Flow Examples

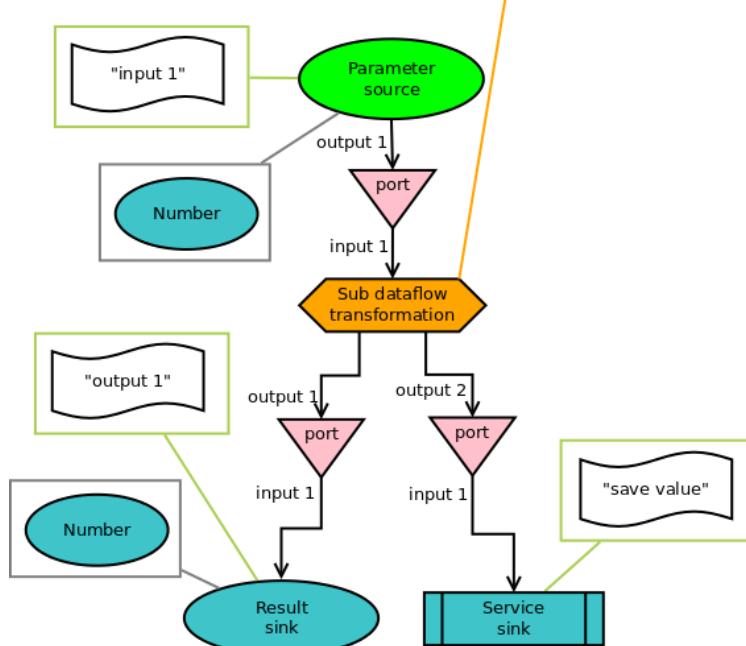
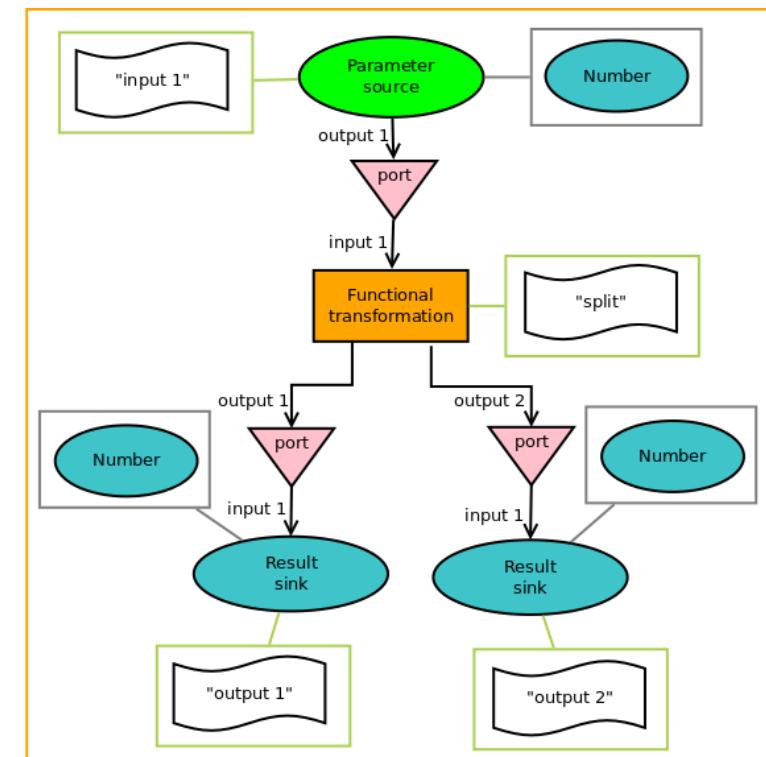
In this section we will take a look at 2 syntactically correct data flow examples.

Notice that the **type** and **identifier** grammar rules are used in the data flow examples in this section. Refer to section 6.2 for a definition of the **type** rule and refer to section 8.1 for a definition of the **Identifier** rule.

The first example has 2 data sources, applies a transformation and writes the transformation output to a sink:



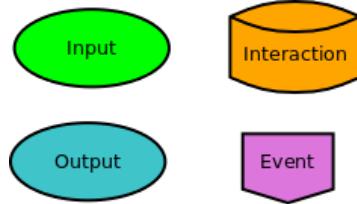
The other example has 1 source, a sub data flow transformation and 2 sinks:



5 Interaction Flow Syntax

5.1 Interaction Flow Units

The units used in the `interaction flow` grammar rule are:



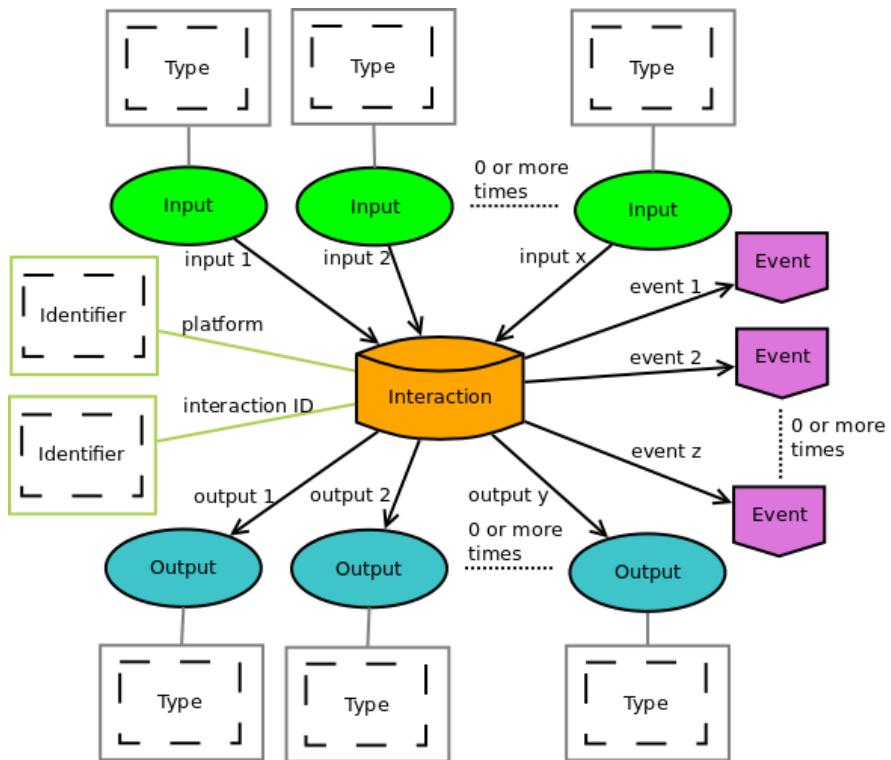
The `input` and `output` units define data input and data output for interactions, respectively. The `event` unit defines control flow events/branches for the interaction. The `interaction` unit is the central point of the `interaction flow` grammar rule, which is defined in the next section 5.2.

5.2 Interaction Flow Grammar Rule

This section defines the `interaction flow` grammar rule:



The `interaction flow` grammar rule has a single expansion which is:



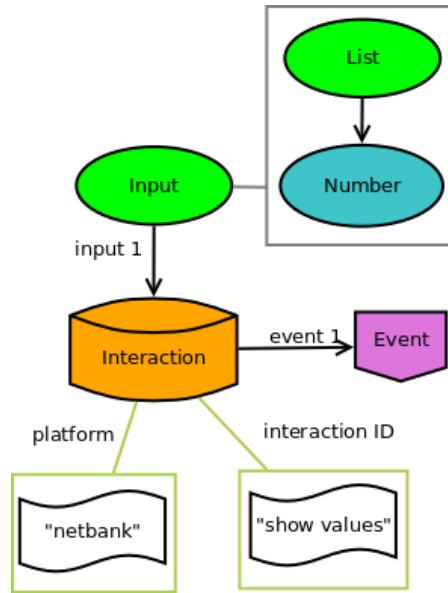
where the `identifier` grammar rule is defined in section 8.1 and the `type` grammar rule is defined in section 6.2.

See section 5.3 for examples of interaction flow syntax.

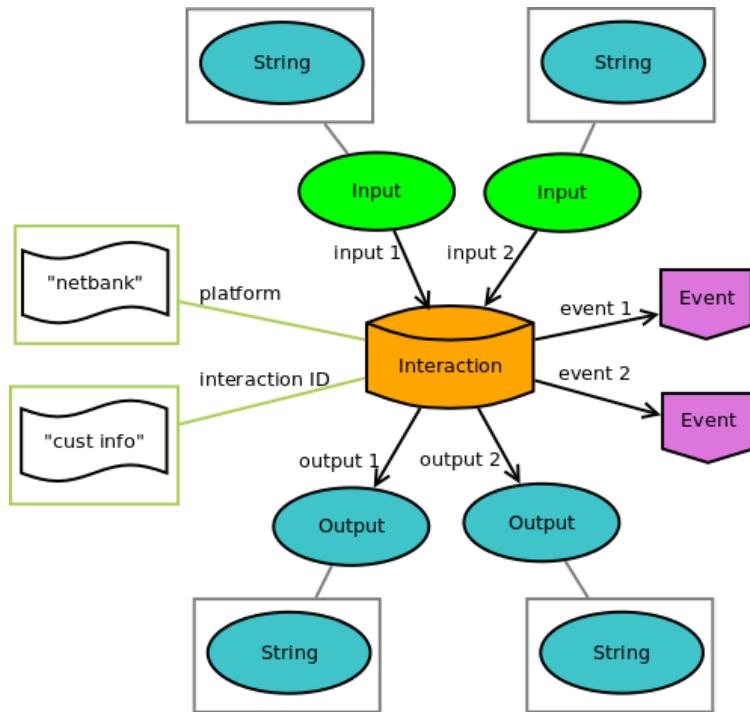
5.3 Interaction Flow Examples

This section contains two examples of interaction flow syntax. Note that the examples use the `identifier` and `type` grammar rules, which are defined in sections 8.1 and 6.2 respectively.

The first example defines an interaction which has a data input and a control flow event:



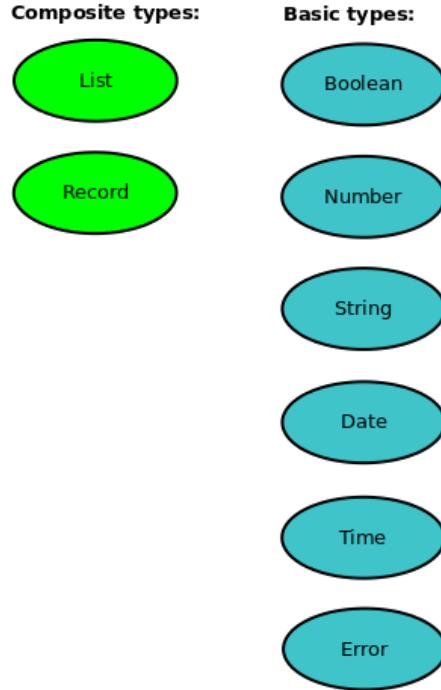
The other example defines an interaction with two data inputs, two data outputs and two control flow events:



6 Type Syntax

6.1 Type Units

The `type` units are distributed into two groups: composite types and basic types:



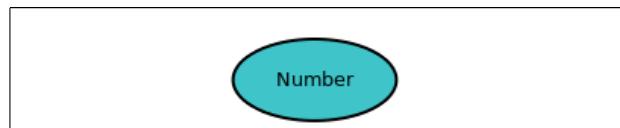
Composite types are types which can be constructed using basic types and other composite types. Basic types are built in.

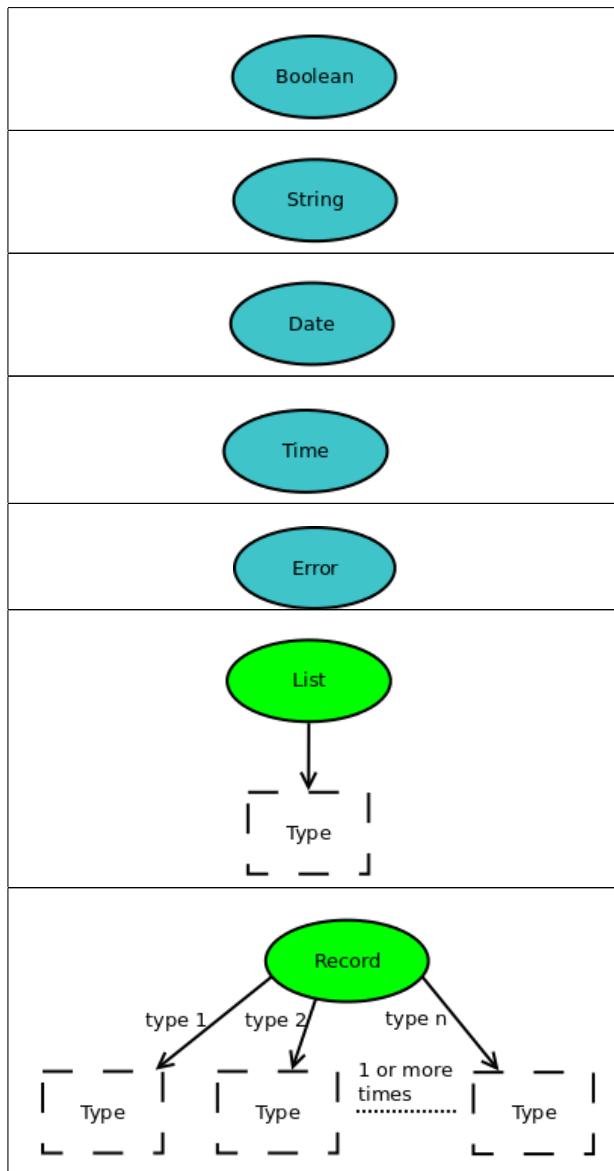
6.2 Type Grammar Rule

This section defines the `type` grammar rule:



The `type` rule uses the type units defined in section 6.1, it has the following expansions:





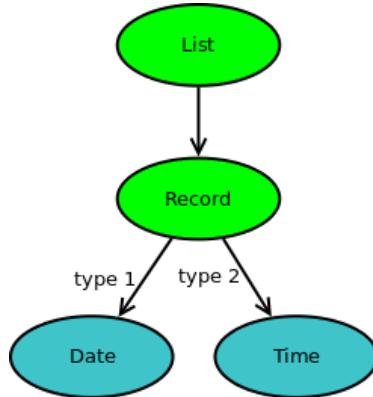
6.3 Type Examples

This section has two examples of syntactically valid types, as defined by the `type` grammar rule in section 6.2.

The first example is a simple number basic type:



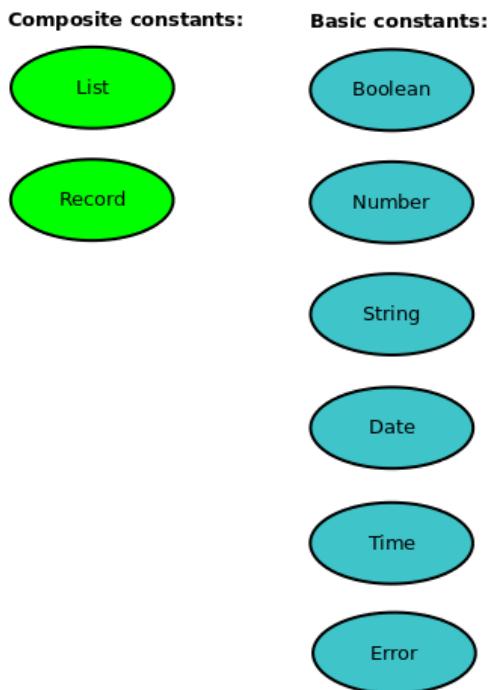
The other syntax example is a table of rows with a date and a time:



7 Constant Syntax

7.1 Constant Units

The **constant** units are the same as the type units:



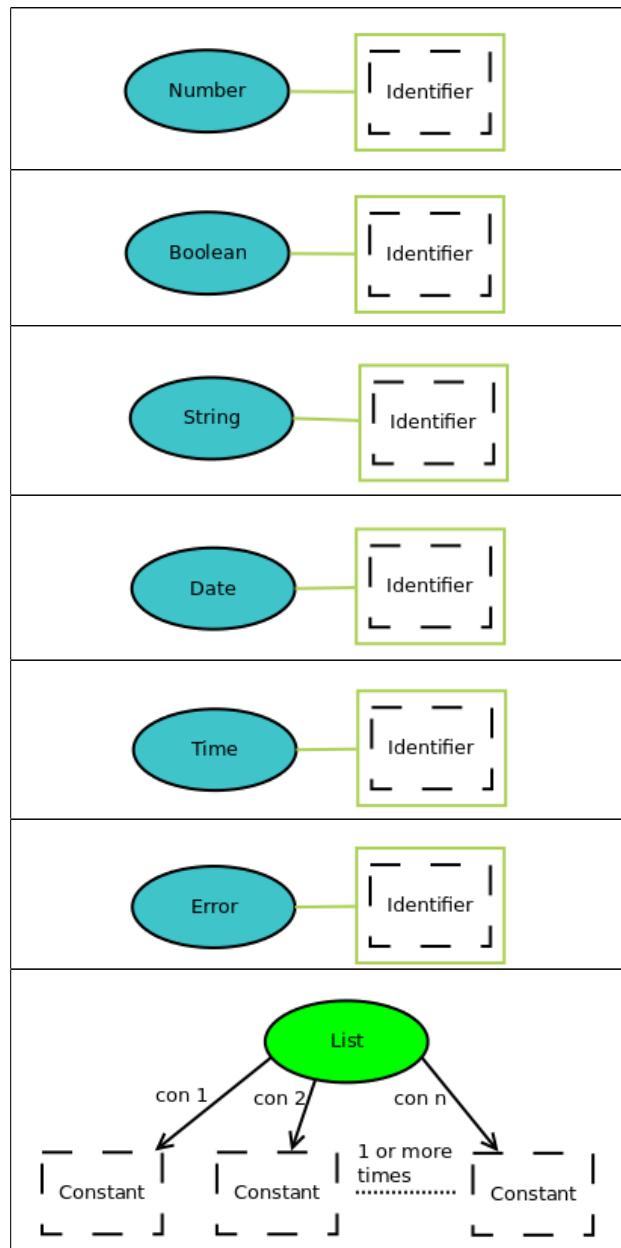
The composite constants are constants that are constructed using basic constants and other composite constants. The basic constants are constructed using an **identifier** to specify the constant value. Composite constants has composite type and basic constants has basic type.

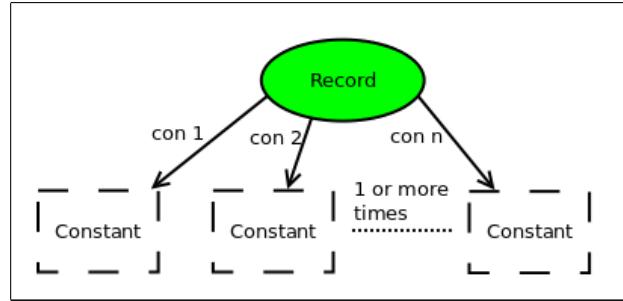
7.2 Constant Grammar Rule

The **constant** grammar rule:



uses the units from section 7.1. It has the following expansions:

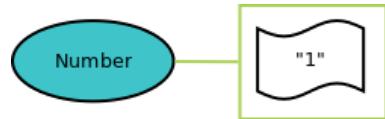




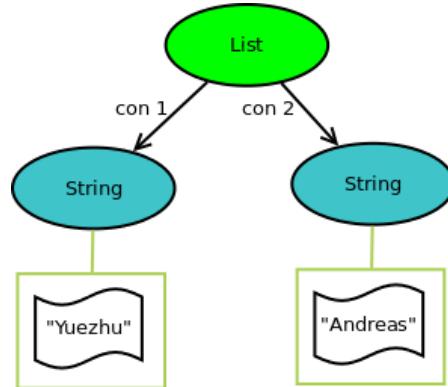
7.3 Constant Examples

This section contains examples of the syntax defined by the `constant` grammar rule. Note that the section uses the `identifier` grammar rule which is defined in section 8.1.

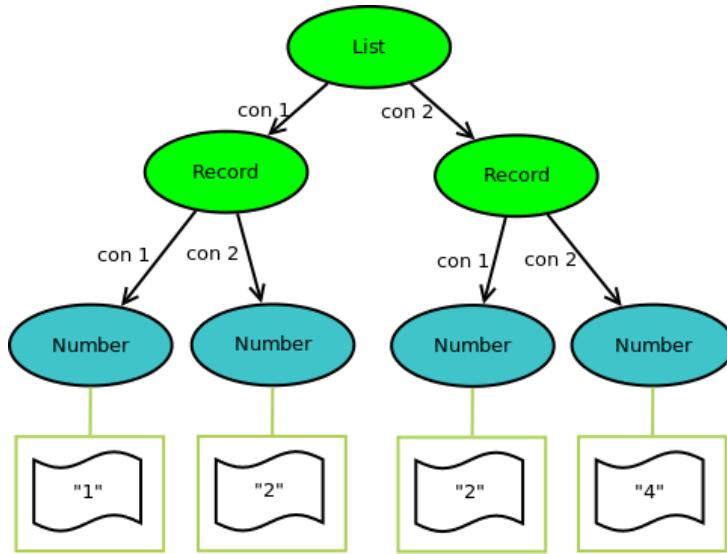
The first example uses the number basic constant:



The next example consists of a table of strings:



The third example is a table of rows with two numbers:



8 Identifier Syntax

8.1 Identifier Grammar Rule

For convenience we will define the **identifier** grammar rule using a regular expression. The **identifier** grammar rule:

$$\boxed{\quad \quad |}$$

Identifier

$$\boxed{\quad \quad }$$

has the expansions:

$$\boxed{[a-zA-Z0-9]+}$$

where the regular expression "[a-zA-Z0-9]+" in the figure must be replaced by a matching text string.

For examples of identifier syntax refer to section 8.2.

8.2 Identifier Examples

In this section we take a look at two identifier syntax examples. For a definition of the syntax used to express identifiers see section 8.1.

In the first example we replace the regular expression "[a-zA-Z0-9]+" with the text string "Hello world"



In the other example we replace the regular expression with " "



9 Program Grammar Rule

The `program` grammar rule defines the syntax of all VOL programs.

The `program` rule:



is defined to have a single expansion which is:

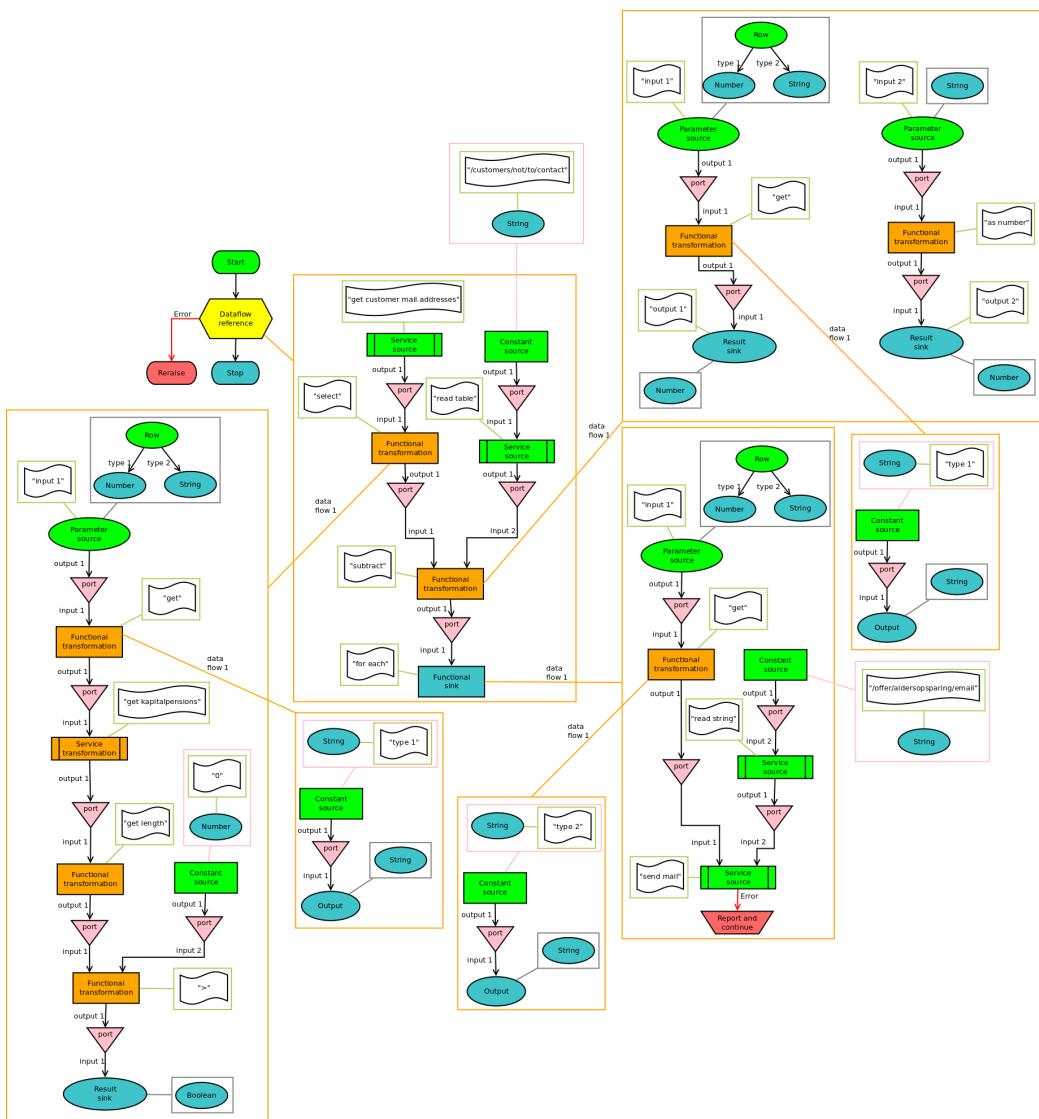


See section 3.2 for the `control flow` grammar rule definition.

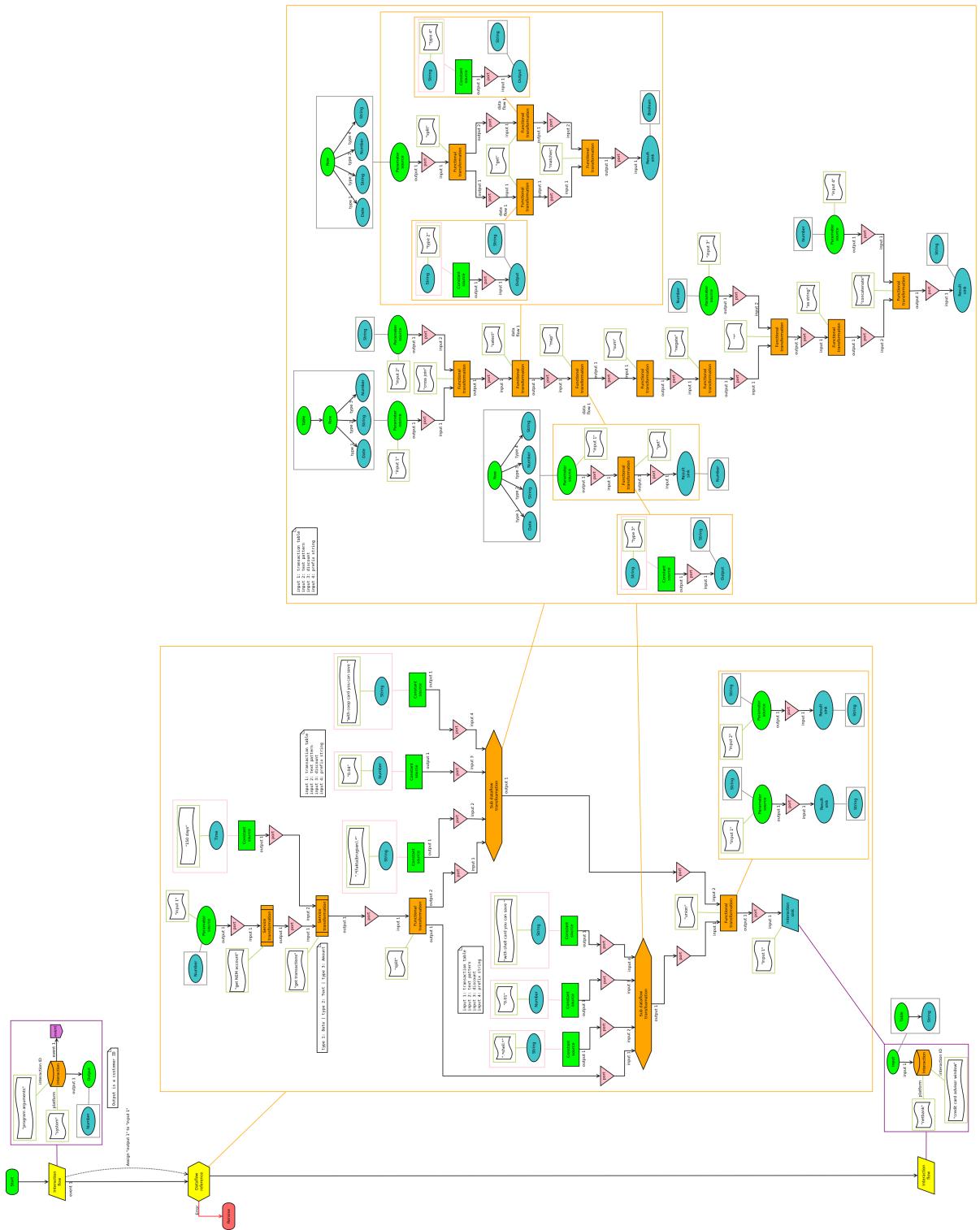
Appendix B

Grammar Examples

B.1 Grammar for US1



B.2 Grammar for US4



Appendix C

Python Tool

```
1 #!/usr/bin/python
2
3 import sys
4 import re
5 import string
6 import subprocess
7 import os
8 from shutil import copyfile
9 import time
10 import psycopg2
11
12 outputDir = "/home/andreas/Dropbox/SDC/HsImpl1/Build/Output/"
13
14 def getInteracts(prg):
15     if not re.findall("\\\{-\\sInteract\\sstart", prg):
16         raise RuntimeError("no start")
17     return map(lambda (x,_): x, re.findall("\\\{-\\sInteract\\s((.|\\n)*?)\\-\\-", prg))
18
19 def parseRecord(rest):
20     assigns = re.findall("\\\{((.|\\n)*?)\\}", rest)[0][0]
21     ret = rest[rest.index("}")+1:].strip()
22     assigns = map(lambda s: s.strip(), assigns.split(","))
23     for i in range(len(assigns)):
24         assigns[i] = tuple(map(lambda s: s.strip(), assigns[i].split("=")))
25     return ret,assigns
26
27 def parseInteract(inter):
28     ret = {}
29     fun,rest = map(lambda s: s.strip(), inter.split(":"))
30     ret["fun"] = fun
31     rest = rest.split(".")
32     ret["platform"] = rest[0].strip()
33     rest = rest[1].split()
34     ret["name"] = rest[0]
35     rest = string.join(rest[1:])
36     rest,args = parseRecord(rest)
37     ret["args"] = args
```

```

38     rest,events = parseRecord(rest)
39     if len(events) == 0:
40         raise RuntimeError("empty event record")
41     ret["events"] = events
42     rest,res = parseRecord(rest)
43     ret["results"] = res
44     return ret
45
46 def recToString(rec):
47     if rec == "" or (len(rec) == 1 and rec[0][0] == ""):
48         return "{}"
49     else:
50         ret = "{}"
51         strs = map(lambda (d,s): "\\\\""+d+"\\\"":\\\""+s+"\\\"", list(rec))
52         for s in strs[:-1]:
53             ret += s+","
54         ret += strs[-1]+"}"
55     return ret
56
57 def recToList(rec):
58     ret = "["
59     if rec != "":
60         strs = map(lambda (d,_): "\\\\""+d+"\\\"", list(rec))
61     if rec != "" and len(strs):
62         for s in strs[:-1]:
63             ret += s+","
64         ret += strs[-1]
65     ret += "]"
66     return ret
67
68 def reportError(e):
69     return "putStrLn (\"{\\\\\"status\\\\\":\\\\\"error\\\\\", \\\\\"result\\\\\":\\\\\"++e++\\\"}\")"
70
71 def genStartInteract(inter, mod):
72     program = "Inter_"+mod+"_"+inter["fun"]
73     fileName = outputDir+"Main_"+mod+"_"+inter["fun"]+".hs"
74     with open(fileName, "w+") as f:
75         f.write("import Internal.Json as J\n")
76         f.write("import System.Environment (getArgs)\n")
77         f.write("import Internal.InteractionRuntime as R\n")
78         f.write("import ServiceError\n")
79         f.write("import System.Exit (exitWith, ExitCode(..))\n")
80         f.write("import "+mod+"\n")
81         f.write("main :: IO ()\n")
82         f.write("main = do\n\t")
83         f.write("args <- getArgs\n\t")
84         f.write("st <- R.anyStartup (R.quote $ args!!2) (R.quote $ args!!3) (R.quote $ args!!0) (R."
85         .quote $ args!!1)\n\t")
86         f.write("case st of {Left ste -> do {"+reportError("J.encodeJsonStr (J.toJson (show ste"
87         ))}+"; exitWith(ExitSuccess)}; Right () -> return ()}\n\t")
88         f.write("if length args == 4 then return () else do {"+reportError("\\\\\"invalid number"
89         of arguments\\\\\"")+"; exitWith (ExitFailure 1)}\n\t")
87         f.write("e <- " + inter["fun"] + "\n\t")
88         f.write("case e of\n\t")
89         f.write("Left e1 -> "+reportError("J.encodeJsonStr (J.toJson (show e1))")+"\n\t\t")

```

```

90         f.write("Right () -> do\n\t\t\t")
91         f.write("r <- R.interactExec (R.quote $ args!!0) (R.quote $ args!!1) (R.quote $ args!!2) (\n"
92             R.quote $ args!!3) \"""+program+"\" as es \"\\\\\""+inter["name"]+"\\\\\" \\"\\\\\""+inter["\n"
93                 platform"]+"\\\\\"\\n\t\t\t")
94         f.write("case r of {Left e2 -> "+reportError("J.encodeJsonStr (J.strToJson (show e2))")+";\n"
95             Right () -> return ()}\n\t\t")
96         f.write("where as = "+recToString(inter["args"])+"\n\t")
97         f.write("      es = "+recToList(inter["events"])+"\n")
98         f.flush()
99     return fileName
100
101
102 def genInteract(inter, mod, imap):
103     fileNames = []
104     if inter["fun"] == "start":
105         fileNames.append(genStartInteract(inter, mod))
106     fn = outputDir+"Inter_"+mod+"_"+inter["fun"]+".hs"
107     fileNames.append(fn)
108     with open(fn, "w+") as f:
109         f.write("import Internal.Json as J\n")
110         f.write("import System.Environment (getArgs)\n")
111         f.write("import Internal.InteractionRuntime as R\n")
112         f.write("import ServiceError\n")
113         f.write("import SeqServiceIO as SIO\n")
114         f.write("import "+mod+"\n")
115         f.write("main :: IO ()\n")
116         f.write("main = do\n\t")
117         f.write("args <- getArgs\n\t")
118         f.write("R.anyStartup (R.quote $ args!!4) (R.quote $ args!!5) (R.quote $ args!!2) (R.quote\n"
119             "$ args!!3)\n\t")
120         f.write("if length args /= 6\n\t\t")
121         f.write("then do\n\t\t")
122         f.write(reportError("\"\\\\\"invalid number of arguments\\\\\"\"")+"\n\t\t")
123         f.write("else do\n\t\t")
124         f.write("s <- SIO.runSeqServiceIO $ R.saveStateRecord (R.quote $ args!!2) (R.quote $ args\n"
125             "!!0) assigns (R.quote $ args!!1)\n\t\t")
126         f.write("case s of\n\t\t\t")
127         f.write("Left err -> do\n\t\t\t")
128         f.write(reportError("J.encodeJsonStr (J.strToJson (show err))")+"\n\t\t\t")
129         f.write("Right () -> do\n\t\t\t")
130         for e in inter["events"]:
131             if e[1] != "terminate":
132                 e1 = e[1]
133                 i = imap[e[1]]
134                 program = "Inter_"+mod+"_"+i["fun"]
135             else:
136                 i = {"args": "", "events": "", "name": "", "platform": ""}
137                 e1 = "R.terminate (R.quote $ args!!2) (R.quote $ args!!3) (R.quote $ args!!4) (R."
138                     quote $ args!!5) \"Main_"+mod+"_start\""
139                 program = "nil"
140                 f.write("if head args == \"\\\\\""+e[0]+"\\\\\"\\n\t\t\t")
141                 f.write("then let {as = "+recToString(i["args"])+"; es = "+recToList(i["events"])+"}\n"
142                     "\t\t\t")
143                 f.write("in "+e1+" >= \\e -> case e of\n\t\t\t")
144                 f.write("Left e1 -> "+reportError("J.encodeJsonStr (J.strToJson (show e1))")+"\n\t\t\t")
145                 f.write("\t\t\t")

```

```

137         f.write("Right () -> R.interactExec (R.quote $ args!!2) (R.quote $ args!!3) (R.quote $ args!!4) (R.quote $ args!!5) \"+program+\" as es \"\\\\\"+i[\"name\"]+\"\\\\\"\\\" \\"\\\\\"+i[\"platform\"]+\"\\\\\"\\\" >>= \\r -> case r of {Left e2 -> "+reportError("J.encodeJsonStr (J. strToJson (show e2)))+"; Right () -> return ()}\n\\t\\t\\t\\t\\t\\t\\t\\t")
138         f.write("else ")
139         f.write(reportError("\\"\\\\\"unexpected interaction event received\\\\\"\\\")+\"\\n\\t")
140         f.write("where assigns = "+recToString(inter["results"])+"\n")
141         f.flush()
142     return fileNames
143
144 def usageExit():
145     print "usage: " + sys.argv[0] + " SOURCEFILE PLATFORM USERGROUP"
146     exit(1)
147
148 if len(sys.argv) != 4:
149     usageExit()
150
151 outputDir += sys.argv[2] + "/"
152 name = sys.argv[1]
153 f = open(name)
154 _,mod1 = os.path.split(name)
155 mod = mod1.split(".") [0]
156
157 prg = f.read()
158 f.close()
159 imap = {}
160 for i in getInteracts(prg):
161     inter = parseInteract(i)
162     imap[inter["fun"]] = inter
163
164 files = []
165 for i in imap.values():
166     files += genInteract(i, mod, imap)
167
168 copyfile(name, outputDir + mod1)
169 os.chdir(outputDir)
170 shpath = outputDir+mod+".sh"
171 shfile = open(shpath, "w")
172 for f in files:
173     cmd = "ghc -dynamic -fno-warn-tabs -XFlexibleContexts -i/home/andreas/Dropbox/SDC/HsImpl1/ - odir . -hidir . " + os.path.split(f)[1]
174     shfile.write("echo "+cmd+"\n")
175     shfile.write("touch "+os.path.split(f)[1]+" || exit 1\n")
176     shfile.write(cmd+" || exit 1\n")
177     shfile.write("sync "+os.path.split(f)[1].split(".")[0]+\n")
178 shfile.close()
179
180 if os.system("sh "+shpath):
181     exit(1)
182
183 conn = psycopg2.connect("dbname='sdc' user='sdc' host='localhost' password='sdc'")
184 cur = conn.cursor()
185 cur.execute("DELETE FROM Programs WHERE program = '"+mod+"'")
186 cur.execute("INSERT INTO Programs VALUES ('"+mod+"', "+sys.argv[3]+", '"+sys.argv[2]+", 'Main_"+mod+"_start')")

```

187 conn.commit();

LISTING C.1: Build/build.py

Appendix D

Haskell Server and Platforms

```
1 module SeqServiceIO (
2     ServiceIO,
3     SeqServiceIO,
4     runSeqServiceIO,
5     errorSIO,
6     parSIO,
7     setVarSIO,
8     getVarSIO,
9     eitherSIO,
10    debugSIO) where
11 import Internal.ServiceIO
```

LISTING D.1: SeqServiceIO.hs

```
1 module ServiceError (ServiceError(..)) where
2
3 import Control.Exception (SomeException)
4
5 {- The ServiceError is used to indicate a transaction failed, and that it has
6 - been rolled back. -}
7 data ServiceError =
8     {- When something does not exist. -}
9     DoesNotExist String |
10    {- A service threw an exception. -}
11    ServiceException SomeException |
12    {- Was unable to encode/decode JSON. -}
13    JSONException String |
14    {- Unable to start program. -}
15    CannotStartup String |
16    {- Invalid argument -}
17    InvalidArgument String |
18    {- Some unexpected error -}
19    InternalFailure String
20
21 instance Show ServiceError where
22     show (DoesNotExist s) = "DoesNotExist - "++s
23     show (ServiceException e) = "ServiceException - "++show e
```

```
24     show (JSONException s) = "JSONException - "++s
25     show (CannotStartup s) = "CannotStartup - "++s
26     show (InvalidArgumentException s) = "InvalidArgumentException - "++s
27     show (InternalFailure s) = "InternalFailure - "++s
```

LISTING D.2: ServiceError.hs

```
1 {-# LANGUAGE OverloadedStrings #-}
2
3 module Runtime (
4     maybeToEither,
5     dataMap,
6     dataFilter,
7     dataJoin,
8     dataForEach,
9     dataAccumulate,
10    dataLength,
11    dataAllTrue,
12    dataAnyTrue,
13    dataSum,
14    dataContains,
15    dataAppend,
16    dataPrepend,
17    dataAugment,
18    dataEqual,
19    dataNotEqual,
20    dataGreater,
21    dataGreaterEqual,
22    dataLess,
23    dataLessEqual,
24    dataAnd,
25    dataOr,
26    dataNot,
27    dataPlus,
28    dataMinus,
29    dataDiv,
30    dataMul,
31    dataNegate,
32    dataStringConcatenate,
33    dataStringContains,
34    dataStringLength,
35    dataStringFormat
36 ) where
37
38 import ServiceError
39 import SeqActIO
40 import Internal.ServiceIO
41 import Control.Monad (filterM)
42 import ServiceError
43 import Internal.Json
44 import SeqServiceIO
45 import Services
46 import Data.Foldable (foldrM)
47 import Data.Scientific
48 import qualified Data.HashMap.Strict as Map
```

```

49 import qualified Data.Text as Text
50 import Data.Functor ((<$>))
51 import Data.List as List
52
53 ----- Generic operations -----
54
55 maybeToEither :: ServiceError -> Maybe a -> Either ServiceError a
56 maybeToEither _ (Just x) = Right x
57 maybeToEither e Nothing = Left e
58
59 dataBinOp :: (Json -> Either ServiceError a) -> (a -> a -> b) -> (Json,Json) -> SeqActIO b
60 dataBinOp cast op (left,right) = do
61     x <- eitherSIO (cast left)
62     y <- eitherSIO (cast right)
63     return (op x y)
64
65 dataUnaryOp :: (Json -> Either ServiceError a) -> (a -> b) -> Json -> SeqActIO b
66 dataUnaryOp cast op arg = eitherSIO (cast arg) >>= return . op
67
68 ----- Functional operations -----
69
70 dataMap :: (Json -> ActID -> SeqActIO Json) -> Json -> ActID -> SeqActIO Json
71 dataMap f input actID = do
72     list <- eitherSIO (jsonToList input)
73     listToJson <$> mapM (flip f actID) list
74
75 dataFilter :: (Json -> ActID -> SeqActIO Json) -> Json -> ActID -> SeqActIO Json
76 dataFilter f input actID = do
77     list <- eitherSIO (jsonToList input)
78     listToJson <$> filterM (\x -> f x actID >>= eitherSIO . jsonToBool) list
79
80 dataJoin :: ((Json,Json) -> ActID -> SeqActIO Json) -> (Json,Json) -> ActID -> SeqActIO Json
81 dataJoin pred (left,right) actID = do
82     leftList <- eitherSIO $ jsonToList left
83     rightList <- eitherSIO $ jsonToList right
84     let leftSelect leftItem acc = let
85         rightSelect rightItem acc = do
86             j <- pred (leftItem,rightItem) actID
87             b <- eitherSIO (jsonToBool j)
88             if b then valueJoin (leftItem,rightItem) >>= return . (:acc)
89                 else return acc
90             in foldrM rightSelect acc rightList
91     listToJson <$> foldrM leftSelect [] leftList
92
93 valueJoin :: (Json,Json) -> SeqActIO Json
94 valueJoin (left,right) = do
95     let left' = asMap left "Left"
96     let right' = asMap right "Right"
97     leftMap <- eitherSIO (jsonToMap left')
98     rightMap <- eitherSIO (jsonToMap right')
99     let ret = mapJoin (Map.empty :: Map.HashMap Text.Text Json) (Map.toList leftMap) rightMap
100    return $ mapToJson ret
101    where asMap :: Json -> Text.Text -> Json
102        asMap j name =
103            if isJsonMap j

```

```

104         then j
105     else let m = Map.empty :: Map.HashMap Text.Text Json
106         m' = Map.insert name j m
107     in mapToJson m'
108   mapJoin acc [] rightMap = Map.union acc rightMap
109   mapJoin acc ((k,vl):tl) rightMap =
110     case Map.lookup k rightMap of
111       Nothing -> mapJoin (Map.insert k vl acc) tl rightMap
112       Just vr -> let
113         acc1 = Map.insert (Text.append "Left_" k) vl acc
114         acc2 = Map.insert (Text.append "Right_" k) vr acc1
115       in mapJoin acc2 tl (Map.delete k rightMap)
116
117 dataForEach :: (Json -> ActID -> SeqActIO ()) -> Json -> ActID -> SeqActIO ()
118 dataForEach f input actID = do
119   list <- eitherSIO (jsonToList input)
120   mapM_ (flip f actID) list
121
122 ----- List operations -----
123
124 dataLength :: Json -> SeqActIO Json
125 dataLength j = do
126   list <- eitherSIO (jsonToList j)
127   return (intToJson $ toInteger (length list))
128
129 dataFoldFromLeft :: ((Json,Json) -> ActID -> SeqActIO Json) -> (Json,Json) -> ActID -> SeqActIO
130   Json
131 dataFoldFromLeft fun (acc,jlist) actID = do
132   list <- eitherSIO (jsonToList jlist)
133   listFold (acc,list)
134   where listFold (acc,[]) = return acc
135     listFold (acc,(x:xs)) = do
136       acc' <- fun (acc,x) actID
137       listFold (acc',xs)
138
139 dataAccumulate :: ((Json,Json) -> ActID -> SeqActIO Json) -> (Json,Json) -> ActID -> SeqActIO Json
140 dataAccumulate fun (list,acc) = dataFoldFromLeft (\(a,b) aid -> fun (b,a) aid) (acc,list)
141
142 dataContains :: (Json,Json) -> ActID -> SeqActIO Json
143 dataContains (js,item) actID = dataFoldFromLeft op (boolToJson False, js) actID
144   where op (acc,x) _ = do
145     b <- eitherSIO (jsonToBool acc)
146     if b then return acc else dataEqual (item,x)
147
148 dataAppend :: (Json,Json) -> SeqActIO Json
149 dataAppend (jlist,item) = do
150   list <- eitherSIO $ jsonToList jlist
151   return $ listToJson (list++[item])
152
153 dataPrepend :: (Json,Json) -> SeqActIO Json
154 dataPrepend (jlist,item) = do
155   list <- eitherSIO $ jsonToList jlist
156   return $ listToJson (item:list)
157
158 dataAugment :: (Json,Json) -> ActID -> SeqActIO Json

```

```

158 dataAugment (left,item) actID = do
159     let right = listToJson [item]
160     dataJoin (\_ _ -> return (boolToJson True)) (left,right) actID
161
162 ----- Boolean operations -----
163
164 dataAnd :: (Json,Json) -> SeqActIO Json
165 dataAnd arg = dataBinOp jsonToBool (&&) arg >>= return . boolToJson
166
167 dataOr :: (Json,Json) -> SeqActIO Json
168 dataOr arg = dataBinOp jsonToBool (||) arg >>= return . boolToJson
169
170 dataNot :: Json -> SeqActIO Json
171 dataNot arg = dataUnaryOp jsonToBool not arg >>= return . boolToJson
172
173 dataAllTrue :: Json -> ActID -> SeqActIO Json
174 dataAllTrue js actID = dataFoldFromLeft (\a _ -> dataAnd a) (boolToJson True, js) actID
175
176 dataAnyTrue :: Json -> ActID -> SeqActIO Json
177 dataAnyTrue js actID = dataFoldFromLeft (\a _ -> dataOr a) (boolToJson False, js) actID
178
179 ----- Numeric operations -----
180
181 dataNumericBinOp :: (Scientific -> Scientific -> Scientific) -> (Json,Json) -> SeqActIO Json
182 dataNumericBinOp op arg = dataBinOp jsonToScientific op arg >>= return . scientificToJson
183
184 dataPlus :: (Json,Json) -> SeqActIO Json
185 dataPlus = dataNumericBinOp (+)
186
187 dataMinus :: (Json,Json) -> SeqActIO Json
188 dataMinus = dataNumericBinOp (-)
189
190 dataMul :: (Json,Json) -> SeqActIO Json
191 dataMul = dataNumericBinOp (*)
192
193 dataNegate :: Json -> SeqActIO Json
194 dataNegate arg = dataUnaryOp jsonToScientific negate arg >>= return . scientificToJson
195
196 dataDiv :: (Json,Json) -> SeqActIO Json
197 dataDiv (jnom,jden) = do
198     snom <- eitherSIO (jsonToScientific jnom)
199     sden <- eitherSIO (jsonToScientific jden)
200     if sden == 0
201         then errorSIO (InvalidArgument "Division by '0'")
202     else
203         let rnom = toRational snom
204             rden = toRational sden
205             Right (r,_) = fromRationalRepetend Nothing (rnom/rden)
206         in return (scientificToJson r)
207
208 dataSum :: Json -> ActID -> SeqActIO Json
209 dataSum js actID = dataFoldFromLeft (\a _ -> dataPlus a) (intToJson 0, js) actID
210
211 ----- Comparison operations -----
212

```

```

213 dataEqual :: (Json,Json) -> SeqActIO Json
214 dataEqual (x,y) = return (boolToJson $ x==y)
215
216 dataNotEqual :: (Json,Json) -> SeqActIO Json
217 dataNotEqual (x,y) = return (boolToJson $ x/=y)
218
219 dataGreater :: (Json,Json) -> SeqActIO Json
220 dataGreater arg = dataBinOp jsonToScientific (>) arg >>= return . boolToJson
221
222 dataGreaterEqual :: (Json,Json) -> SeqActIO Json
223 dataGreaterEqual arg = dataBinOp jsonToScientific (>=) arg >>= return . boolToJson
224
225 dataLess :: (Json,Json) -> SeqActIO Json
226 dataLess arg = dataBinOp jsonToScientific (<) arg >>= return . boolToJson
227
228 dataLessEqual :: (Json,Json) -> SeqActIO Json
229 dataLessEqual arg = dataBinOp jsonToScientific (<=) arg >>= return . boolToJson
230
231 ----- String operations -----
232
233 dataStringConcatenate :: (Json,Json) -> SeqActIO Json
234 dataStringConcatenate arg = dataBinOp jsonToStr (++) arg >>= return . strToJson
235
236 dataStringContains :: (Json,Json) -> SeqActIO Json
237 dataStringContains arg = dataBinOp jsonToStr (flip List.isInfixOf) arg >>= return . boolToJson
238
239 dataStringLength :: Json -> SeqActIO Json
240 dataStringLength arg = dataUnaryOp jsonToStr length arg >>= return . intToJson . toInteger
241
242 dataStringFormat :: String -> [Json] -> SeqActIO Json
243 dataStringFormat fmt args = do
244     f <- stringFormat fmt args
245     return (strToJson f)
246     where stringFormat fmt [] = return fmt
247         stringFormat ('%'':':n':fmt) (j:js) = do
248             n <- eitherSIO (jsonToScientific j)
249             fmap (show n++) (stringFormat fmt js)
250         stringFormat ('%'':':b':fmt) (j:js) = do
251             b <- eitherSIO (jsonToBool j)
252             fmap (show b++) (stringFormat fmt js)
253         stringFormat ('%'':':s':fmt) (j:js) = do
254             s <- eitherSIO (jsonToStr j)
255             fmap (s++) (stringFormat fmt js)
256         stringFormat ('%'':':%':fmt) js =
257             fmap ('%'':') (stringFormat fmt js)
258         stringFormat (c:fmt) js =
259             fmap (c:) (stringFormat fmt js)

```

LISTING D.3: Runtime.hs

```

1 {-# LANGUAGE OverloadedStrings, FlexibleContexts #-}
2
3 module Main (main) where
4
5 import Prelude as P

```

```

6 import qualified Data.Text as T
7 import Web.Spock.Safe
8 import Internal.Database
9 import Internal.Json
10 import Database.HDBC as DB
11 import Control.Monad.IO.Class (MonadIO, liftIO)
12 import ServiceError
13 import System.IO
14 import Internal.InteractionRuntime (InterArgs(..))
15 import qualified Data.ByteString.Char8 as BC
16 import qualified Network.Http.Client as C
17 import qualified System.IO.Streams.Internal as IOI
18 import qualified Data.Map as M
19 import PlatformInteractions
20
21 main :: IO ()
22 main = runSpock 5001 . spockT id $ do
23     get ("Medarbejderportal" <//> var) runP1
24     get ("Netbank" <//> var) runP2
25     get ("Medarbejderportal/show" <//> var) $ showPlatformFile "Medarbejderportal"
26     get ("Netbank/show" <//> var) $ showPlatformFile "Netbank"
27     get ("jquery.js") $ showFile (buildDirectory++"jquery-1.12.2.min.js")
28     post ("Medarbejderportal/make/interaction") $ makeInteraction "Medarbejderportal"
29     post ("Netbank/make/interaction") $ makeInteraction "Netbank"
30     post ("Medarbejderportal/running/program") $ queryRunningProgram "Medarbejderportal"
31     post ("Netbank/running/program") $ queryRunningProgram "Netbank"
32
33 buildDirectory :: String
34 buildDirectory = "/home/andreas/Dropbox/SDC/HsImpl1/Build/"
35
36 runP1 :: (MonadIO m) => String -> ActionCtxT ctx m ()
37 runP1 username = runP "#FFEEDD" "\\\"Medarbejderportal\\\"" username
38
39 runP2 :: (MonadIO m) => String -> ActionCtxT ctx m ()
40 runP2 username = runP "#E6E6FA" "\\\"Netbank\\\"" username
41
42 getDbConn :: (MonadIO m) => ActionCtxT ctx m Conn
43 getDbConn = do
44     c1 <- liftIO $ dbConnect
45     let conn = case c1 of
46         Left e -> error (show e)
47         Right c -> c
48     return conn
49
50 runP :: (MonadIO m) => String -> String -> String -> ActionCtxT ctx m ()
51 runP bgcolor platform username = do
52     liftIO $ putStrLn "platform: start login"
53     conn <- getDbConn
54     r <- liftIO $ DB.quickQuery conn logonQ [
55         DB.toSql dbuser, DB.toSql platform]
56     liftIO $ putStrLn "platform: finish login"
57     if length r == 0
58         then html (T.pack doesNotExistHtml)
59         else welcomeScreen conn bgcolor platform username dbuser (
60             DB.fromSql $ P.head (P.head r))

```

```

61    liftIO $ DB.disconnect conn
62    where logonQ = "SELECT usergroup FROM Users " ++
63          "WHERE username = ? and platform = ?"
64    doesNotExistHtml = "<html><body bgcolor=""++bgcolor++">" ++
65          "<h2>user "++username++" does not exist</h2>" ++
66          "</body></html>"
67    dbuser = "\""++username++"\\""
68
69 welcomeScreen :: (MonadIO m) => Conn -> String -> String -> String ->
70     Integer -> ActionCtxT ctx m ()
71 welcomeScreen conn bgcolor platform username dbuser usergroup = do
72     body <- genBody conn platform dbuser usergroup
73     let welcomeHtml = "<html>" ++
74         "<script src='/jquery.js'></script>" ++
75         "<body bgcolor=""++bgcolor++">" ++
76         "<h2>hello "++username++"</h2>" ++ body ++
77         "</body></html>"
78     html (T.pack welcomeHtml)
79
80 genBody :: (MonadIO m) => Conn -> String -> String -> Integer ->
81     ActionCtxT ctx m String
82 genBody conn platform username usergroup = do
83     newTasks <- getNewTaskList platform usergroup
84     let newTasksHtml = getNewTaskHtml username platform newTasks
85     waitingTasksList <- getWaitingTaskList platform username
86     let waitingTasksHtml = getWaitingTaskHtml waitingTasksList
87     failedTasksList <- getFailedTaskList platform username
88     let failedTasksHtml = getFailedTaskHtml platform username failedTasksList
89     runningTasksList <- getRunningTaskList conn platform username
90     let runningTasksHtml = getRunningTaskHtml platform username runningTasksList
91     terminatedTasksList <- getTerminatedTaskList platform username
92     let terminatedTasksHtml = getTerminatedTaskHtml platform username terminatedTasksList
93     let contents = "<h2>new tasks</h2>" ++
94         "<p><ul>" ++
95         newTasksHtml ++
96         "</ul></p>" ++
97
98         "<h2>waiting tasks</h2>" ++
99         "<p><ul>" ++
100        waitingTasksHtml ++
101        "</ul></p>" ++
102
103        "<h2>successfully finished tasks</h2>" ++
104        "<p><ul>" ++
105        terminatedTasksHtml ++
106        "</ul></p>" ++
107
108        "<h2>failed tasks</h2>" ++
109        "<p><ul>" ++
110        failedTasksHtml ++
111        "</ul></p>" ++
112
113        "<h2>running tasks</h2>" ++
114        "<p><ul>" ++
115        runningTasksHtml ++

```

```
116      "</ul></p>" ++
117
118      "<script>" ++
119      "function newTaskSubmit(p) {" ++
120      "var v = document.getElementById('newIdV'+p).value;" ++
121      "if(!(/^[a-zA-Z0-9]+/.test(v))){" ++
122      "alert('invalid task identifier');" ++
123      "return false;}" ++
124      "document.getElementById('newIdH'+p).value = '\\"'+v+'\\"';" ++
125      "document.getElementById('idSubmit'+p).value = '\\"'+v+'\\"';" ++
126      "var f = $('#newTaskForm'+p);" ++
127
128      "$.ajax({" ++
129      "type: 'POST'," ++
130      "url: 'http://localhost:5005'," ++
131      "dataType: 'text'," ++
132      "crossDomain: true," ++
133      "data: f.serialize()," ++
134      "success: function(msg) {" ++
135      "if (msg.startsWith('ok')) {" ++
136      "document.getElementById('newTaskSubmitForm'+p).submit();" ++
137      "}else{" ++
138      "alert(msg);" ++
139      "}" ++
140      "}, " ++
141      "error: function(msg) {" ++
142      "alert('sorry, internal failure');" ++
143      "}" ++
144      "});" ++
145
146      "return false;" ++
147      "}" ++
148
149      "function getRunningTaskSubmit(p) {" ++
150      "document.getElementById('getRunningTaskForm'+p).submit();" ++
151      "return false;" ++
152      "}" ++
153
154      "function clearTaskSubmit(form,url) {" ++
155      "var f = $('#'+form);" ++
156
157      "$.ajax({" ++
158      "type: 'POST'," ++
159      "url: 'http://localhost:5005'," ++
160      "dataType: 'text'," ++
161      "crossDomain: true," ++
162      "data: f.serialize()," ++
163      "success: function(msg) {" ++
164      "if (msg.startsWith('ok')) {" ++
165      "location.reload();" ++
166      "}else{" ++
167      "alert(msg);" ++
168      "}" ++
169      "}, " ++
170      "error: function(msg) {" ++
```

```

171         "alert('sorry, internal failure');" ++
172     "}" ++
173   "});" ++
174
175   "return false;" ++
176 "}" ++
177 "</script>"
178 return contents
179
180 getNewTaskList :: (MonadIO m) => String -> Integer ->
181     ActionCtxT ctx m [(String, String)]
182 getNewTaskList platform group = do
183     liftIO $ putStrLn "platform: get new task list from server"
184     liftIO $ C.postForm mainServer ntArgs $ \_ s -> do
185         bs <- IOI.read s
186         putStrLn $ "got new task list response: " ++ show bs
187         case bs of
188             Nothing -> error "unexpected response"
189             Just bs1 -> return $ read (drop 2 (BC.unpack bs1))
190         where ntArgs = [("action", "\listnew"),
191                         ("group", BC.pack $ show group),
192                         ("platform", BC.pack $ platform)]
193
194 removeQuotes :: String -> String
195 removeQuotes = filter ('"'/=)
196
197 getNewTaskHtml :: String -> String -> [(String, String)] -> String
198 getNewTaskHtml dbuser platform list = concat . flip map list $ \(program,_) ->
199     let p = removeQuotes program in
200     "<li>" ++
201     "<form id=\\"newTaskSubmitForm"++p++"\\" method=\\"POST\\" " ++
202         "action='/"++removeQuotes platform++"/running/program'" ++
203         "<input type=\\"hidden\\" name=\\"startuser\\" value='"++dbuser++"'/>" ++
204         "<input type=\\"hidden\\" name=\\"startplatform\\" value='"++platform++"'/>" ++
205         "<input type=\\"hidden\\" name=\\"program\\" value='"++program++"'/>" ++
206         "<input type=\\"hidden\\" name=\\"user\\" value='"++dbuser++"'/>" ++
207         "<input type=\\"hidden\\" id=\\"idSubmit"++p++"\\" name=\\"id\\" value='' />" ++
208     "</form>" ++
209     "<form id=\\"newTaskForm"++p++"\\" method=\\"POST\\" >" ++
210         "<input type=\\"hidden\\" name=\\"action\\" value='\\new'" />" ++
211         "<input type=\\"hidden\\" name=\\"program\\" value='"++program++"' />" ++
212         "<input type=\\"hidden\\" name=\\"user\\" value='"++dbuser++"' />" ++
213         "<input type=\\"hidden\\" name=\\"platform\\" value='"++platform++"' />" ++
214         "<input type=\\"hidden\\" name=\\"id\\" id=\\"newIdH"++p++"\\" />" ++
215         "<a style=\\"color:#0000FF\\" href=\\"javascript:{}\\" " ++
216             "onclick=\\"return newTaskSubmit('"++p++"';" ++ "\\">" ++
217             "start" ++
218         "</a> "++p++" # "++
219         "<label for=\\"__id\\"> </label>" ++
220         "<input type=\\"text\\" name=\\"__id\\" id=\\"newIdV"++p++"\\" />" ++
221     "</form>" ++
222     "</li>"
223
224 getWaitingTaskHtml :: [(String, String, String, String, String)] -> String
225 getWaitingTaskHtml list = concat . flip map list $ \ (ident, prog, plat, user, url) ->

```

```

226     "<li>" ++
227     "open <a style=\"color:#0000FF\" href='++removeQuotes url++'>" ++
228         removeQuotes prog++" # "++removeQuotes ident++"</a>" ++
229     " (started by "++removeQuotes user++" on "++removeQuotes plat++)" ++
230     "</li>"
231
232 getWaitingTaskList :: (MonadIO m) => String -> String ->
233     ActionCtxT ctx m [(String, String, String, String, String)]
234 getWaitingTaskList platform user = do
235     liftIO $ putStrLn "platform: get waiting task list from server"
236     liftIO $ C.postForm mainServer wArgs $ \_ s -> do
237         bs <- IOI.read s
238         putStrLn $ "got waiting task list response: " ++ show bs
239         case bs of
240             Nothing -> error "unexpected response"
241             Just bs1 -> return $ read (drop 2 (BC.unpack bs1))
242     where wArgs = [("action", "\"listwaiting\""),
243                    ("user", BC.pack user),
244                    ("platform", BC.pack platform)]
245
246 getFailedTaskHtml :: String -> String -> [(String, String, String)] -> String
247 getFailedTaskHtml platform user list = concat . flip map list $ \(ident, prog, msg) ->
248     let p = removeQuotes prog
249         i = removeQuotes ident in
250     "<li>" ++
251     "<form id=\"getFinishedTaskForm\"++p++i++\" method=\"POST\" " ++
252         "action='http://localhost:5001/"++plat++"/running/program'>" ++
253         "<input type=\"hidden\" name=\"action\" value='\"clearprogram\"' />" ++
254         "<input type=\"hidden\" name=\"user\" value='\"++user++\"' />" ++
255         "<input type=\"hidden\" name=\"platform\" value='\"++platform++\"' />" ++
256         "<input type=\"hidden\" name=\"id\" value='\"++ident++\"' />" ++
257         "<input type=\"hidden\" name=\"program\" value='\"++prog++\"' />" ++
258         "<a style=\"color:#0000FF\" href=\"javascript:{}\" " ++
259             "onclick=\"return clearTaskSubmit('getFinishedTaskForm"+++p++i++");\"++\">" ++
260             "clear" ++
261         "</a> "++p++" # "++i++" : "++msg ++
262     "</form>" ++
263     "</li>"
264     where plat = removeQuotes platform
265
266 getTerminatedTaskHtml :: String -> String -> [(String, String)] -> String
267 getTerminatedTaskHtml platform user list = concat . flip map list $ \ (ident, prog) ->
268     let p = removeQuotes prog
269         i = removeQuotes ident in
270     "<li>" ++
271     "<form id=\"getTerminatedTaskForm\"++p++i++\" method=\"POST\" " ++
272         "action='http://localhost:5001/"++plat++"/running/program'>" ++
273         "<input type=\"hidden\" name=\"action\" value='\"clearprogram\"' />" ++
274         "<input type=\"hidden\" name=\"user\" value='\"++user++\"' />" ++
275         "<input type=\"hidden\" name=\"platform\" value='\"++platform++\"' />" ++
276         "<input type=\"hidden\" name=\"id\" value='\"++ident++\"' />" ++
277         "<input type=\"hidden\" name=\"program\" value='\"++prog++\"' />" ++
278         "<a style=\"color:#0000FF\" href=\"javascript:{}\" " ++
279             "onclick=\"return clearTaskSubmit('getTerminatedTaskForm"+++p++i++");\"++\">" ++
280             "clear" ++

```

```

281     "</a> "++p++" # "++i++
282     "</form>" ++
283     "</li>" ++
284     where plat = removeQuotes platform
285
286 getFailedTaskList :: (MonadIO m) => String -> String ->
287     ActionCtxT ctx m [(String, String, String)]
288 getFailedTaskList platform username = do
289     liftIO $ putStrLn "platform: get failed task list from server"
290     liftIO $ C.postForm mainServer fArgs $ \_ s -> do
291         bs <- IOI.read s
292         putStrLn $ "got failed task list response: " ++ show bs
293         case bs of
294             Nothing -> error "unexpected response"
295             Just bs1 -> return $ read (drop 2 (BC.unpack bs1))
296         where fArgs = [("action", "\"listfailed\""),
297                         ("user", BC.pack username),
298                         ("platform", BC.pack platform)]
299
300 getTerminatedTaskList :: (MonadIO m) => String -> String ->
301     ActionCtxT ctx m [(String, String)]
302 getTerminatedTaskList platform username = do
303     liftIO $ putStrLn "platform: get terminated task list from server"
304     liftIO $ C.postForm mainServer fArgs $ \_ s -> do
305         bs <- IOI.read s
306         putStrLn $ "got terminated task list response: " ++ show bs
307         case bs of
308             Nothing -> error "unexpected response"
309             Just bs1 -> return $ read (drop 2 (BC.unpack bs1))
310         where fArgs = [("action", "\"listterminated\""),
311                         ("user", BC.pack username),
312                         ("platform", BC.pack platform)]
313
314 getRunningTaskHtml :: String -> String -> [(String, String)] -> String
315 getRunningTaskHtml platform user list = concat . flip map list $ \ (ident, prog) ->
316     let p = removeQuotes prog
317         i = removeQuotes ident in
318         "<li>" ++
319         "<form id=\"getRunningTaskForm\"++p++i++\" method=\"POST\" " ++
320             "action='http://localhost:5001/"++plat++"/running/program'>" ++
321             "<input type=\"hidden\" name=\"startuser\" value='++user++' />" ++
322             "<input type=\"hidden\" name=\"startplatform\" value='++platform++' />" ++
323             "<input type=\"hidden\" name=\"user\" value='++user++' />" ++
324             "<input type=\"hidden\" name=\"id\" value='++ident++' />" ++
325             "<input type=\"hidden\" name=\"program\" value='++prog++' />" ++
326             "<a style=\"color:#0000FF\" href=\"javascript:{}\" " ++
327                 "onclick=\"return getRunningTaskSubmit('++p++i++');\"++\">\n" ++
328                 p++" # "++i ++
329             "</a>" ++
330             "</form>" ++
331             "</li>" ++
332             where plat = removeQuotes platform
333
334 getRunningTaskList :: (MonadIO m) => Conn -> String -> String ->
335     ActionCtxT ctx m [(String, String)]

```

```

336 getRunningTaskList conn platform username = do
337     liftIO $ putStrLn "platform: get running task list from server"
338     liftIO $ C.postForm mainServer rArgs $ \_ s -> do
339         bs <- IOI.read s
340         putStrLn $ "got running task list response: " ++ show bs
341         case bs of
342             Nothing -> error "unexpected response"
343             Just bs1 -> return $ read (drop 2 (BC.unpack bs1))
344     where rArgs = [("action", "\listrunning\""),
345                   ("user", BC.pack username),
346                   ("platform", BC.pack platform)]
347
348 startsWithOk :: String -> Bool
349 startsWithOk [] = False
350 startsWithOk [_] = False
351 startsWithOk ('o':':k':_) = True
352 startsWithOk _ = False
353
354 showPlatformFile :: (MonadIO m) -> String -> String -> ActionCtxT ctx m ()
355 showPlatformFile platform f = showFile path
356     where path = buildDirectory++"/Output/"++platform++"/"+f
357
358 showFile :: (MonadIO m) -> String -> ActionCtxT ctx m ()
359 showFile file = do
360     contents <- liftIO $ readFile file
361     html (T.pack contents)
362
363 getArg :: (MonadIO m) -> String -> ActionCtxT ctx m String
364 getArg p = do
365     t <- param (T.pack p)
366     case t of
367         Nothing -> error $ "unable to find argument " ++ p
368         Just a -> return a
369
370 data RunningProgramState =
371     ProgramStateRunning |
372     ProgramStateFailed |
373     ProgramStateWaiting String |
374     ProgramStateTerminated
375     deriving (Read)
376
377 queryRunningProgram :: (MonadIO m) -> String -> ActionCtxT ctx m ()
378 queryRunningProgram platform = do
379     liftIO $ putStrLn ("platform: "+platform++" query running program")
380     user <- getArg "user"
381     ident <- getArg "id"
382     prog <- getArg "program"
383     startuser <- getArg "startuser"
384     startplatform <- getArg "startplatform"
385     let args = [("action", "\querystate\""),
386                 ("startuser", BC.pack startuser),
387                 ("startplatform", BC.pack startplatform),
388                 ("user", BC.pack user),
389                 ("platform", BC.pack ("\\"++platform++"\\")),
390                 ("id", BC.pack ident),

```

```

391         ("program", BC.pack prog)]
392     s <- liftIO $ C.postForm mainServer args $ \_ s -> do
393         bs <- IOI.read s
394         putStrLn $ "got response for running program: " ++ show bs
395         case bs of
396             Nothing -> error "unexpected response"
397             Just bs1 -> return $ (read (drop 2 (BC.unpack bs1))) :: RunningProgramState)
398     let stateRunningHtml = "<body>" ++
399                     "<div id='bodycontents'>" ++
400                     "<form id=\"form\" method=\"POST\" " ++
401                     "action='#'>" ++
402                     "<input type=\"hidden\" name=\"startuser\" value=\"++startuser++\"/>" ++
403                     "<input type=\"hidden\" name=\"startplatform\" value=\"++startplatform" ++
404                     "++\" />" ++
405                     "<input type=\"hidden\" name=\"user\" value=\"++user++\"/>" ++
406                     "<input type=\"hidden\" name=\"id\" value=\"++ident++\"/>" ++
407                     "<input type=\"hidden\" name=\"program\" value=\"++prog++\"/>" ++
408                     "</form>" ++
409                     homeButton platform user ++
410                     "<p>" ++
411                     "task "++removeQuotes prog++" # " ++
412                     removeQuotes ident++" is running" ++
413                     "</p>" ++
414
415                     "<script src='/jquery.js'></script>" ++
416                     "<script>" ++
417
418                     "var interval = setInterval(function() {" ++
419
420                     "$.ajax({" ++
421                     "type: 'POST'," ++
422                     "url: 'http://localhost:5001/" ++
423                     platform++"/running/program'," ++
424                     "data: $('#form').serialize()," ++
425                     "success: function(msg) {" ++
426                     "if (!msg.startsWith(" ++
427                     "\">\n<body><div id='bodycontents'>\"){" ++
428                     "$('#bodycontents').html(msg);" ++
429                     "clearInterval(interval);" ++
430                     "}" ++
431                     "}, " ++
432                     "error: function(msg) {" ++
433                     "$('#bodycontents').html('');" ++
434                     "}" ++
435                     "});" ++
436
437                     "}, 200);" ++
438
439                     "</script>" ++
440
441                     "</div>" ++
442                     "</body>" ++
443     let stateFailedHtml = "<body>" ++
444                     homeButton platform user ++

```

```

444      "<p>" ++
445      "task "++removeQuotes prog++" # " ++
446      removeQuotes ident++" has failed" ++
447      "</p>" ++
448      "</body>" ++
449      let stateWaitingHtml url = "<script>" ++
450          "window.location.replace(\""+url+"\");" ++
451          "</script>" ++
452      let stateTerminatedHtml = "<body>" ++ homeButton platform user ++
453          "<p>" ++
454          "task "++removeQuotes prog++" # " ++
455          removeQuotes ident++" has finished" ++
456          "</p>" ++
457          "</body>" ++
458      case s of
459          ProgramStateRunning -> html (T.pack stateRunningHtml)
460          ProgramStateFailed -> html (T.pack stateFailedHtml)
461          ProgramStateWaiting url -> html (T.pack $ stateWaitingHtml url)
462          ProgramStateTerminated -> html (T.pack stateTerminatedHtml)

```

LISTING D.4: Platforms.hs

```

1 module Json (module Internal.Json) where
2 import Internal.Json

```

LISTING D.5: Json.hs

```

1 {-# LANGUAGE OverloadedStrings, ScopedTypeVariables #-}
2
3 module Main (main) where
4
5 import Control.Monad.IO.Class (MonadIO, liftIO)
6 import qualified Data.Text as T
7 import Web.Spock.Safe
8 import System.Process
9 import Internal.Database
10 import Control.Monad.Trans.Either
11 import ServiceError
12 import Internal.Json
13 import Text.Read (readMaybe)
14 import Control.Monad.Trans.Class (lift)
15 import System.IO (hGetContents)
16 import System.Exit (ExitCode(..))
17 import qualified Database.HDBC as DB
18 import qualified Internal.InteractionRuntime as R
19 import Control.Concurrent (forkIO)
20 import Control.Exception (catch, SomeException)
21 import Data.List (intercalate)
22 import Data.String.Utils (replace)
23 import qualified Data.ByteString.Char8 as C8
24 import qualified Data.ByteString as B
25 import Data.String.Utils (split)
26 import System.Process (shell, createProcess, waitForProcess)
27 import System.Exit (ExitCode(ExitSuccess))

```

```

28 import Control.Concurrent (threadDelay)
29
30 data RunningProgramState =
31     ProgramStateRunning |
32     ProgramStateFailed |
33     ProgramStateWaiting String |
34     ProgramStateTerminated
35     deriving (Read)
36
37 maybeError :: ServiceError -> Maybe a -> Either ServiceError a
38 maybeError e x = case x of
39     Nothing -> Left e
40     Just x1 -> Right x1
41
42 getArg :: (MonadIO m) => String -> ActionCtxT ctx m (Either ServiceError String)
43 getArg p = do
44     t <- param (T.pack p)
45     return $ maybeError (InvalidArgument $ "no \"++p++\" argument") t
46
47 main :: IO ()
48 main = runSpock 5005 . spockT id $ do
49     post root rootAction
50     post ("Hs" </> var) fileAction
51
52 rootAction :: (MonadIO m) => ActionCtxT ctx m ()
53 rootAction = do
54     setHeader "Access-Control-Allow-Origin" "http://localhost:5001"
55     r <- start
56     case r of
57         Left err -> text (T.pack (show err))
58         Right msg -> text (T.pack ("ok " ++ msg))
59
60 fileDestination :: String
61 fileDestination = "/home/andreas/Dropbox/SDC/HsImpl1/Build/"
62
63 buildFile :: String
64 buildFile = fileDestination++"build.py"
65
66 fileAction :: (MonadIO m) => String -> ActionCtxT ctx m ()
67 fileAction code = do
68     b <- body
69     liftIO $ putStrLn ("server: create file " ++ file)
70     liftIO $ B.writeFile file b
71     (_,...,proc) <- liftIO $ createProcess (shell (buildFile++" ++file++" ++plat++" ++group"))
72     exitCode <- liftIO $ waitForProcess proc
73     if (exitCode == ExitSuccess)
74         then do
75             liftIO $ putStrLn ("server: compilation of "++file++" succeeded")
76             text "ok"
77         else do
78             liftIO $ putStrLn ("server: compilation of "++file++" failed")
79             text "Error: server failed compiling program"
80     where sp = split "." code
81         group = sp!!2
82         plat = sp!!1

```

```

83         file = fileDestination ++ plat ++ "/" ++ sp!!0 ++ ".hs"
84
85 start :: (MonadIO m) => ActionCtxT ctx m (Either ServiceError String)
86 start = runEitherT $ do
87     action <- lift (getArg "action") >>= hoistEither
88     lift (startAction action) >>= hoistEither
89
90 startAction :: (MonadIO m) => String -> ActionCtxT ctx m (Either ServiceError String)
91 startAction action
92     | action == "\"new\"" = newProgram
93     | action == "\"continue\"" = continueProgram
94     | action == "\"fail\"" = failProgram
95     | action == "\"waiting\"" = addWaitingProgram
96     | action == "\"listwaiting\"" = getWaitingTaskList
97     | action == "\"listfailed\"" = getFailedTaskList
98     | action == "\"listterminated\"" = getTerminatedTaskList
99     | action == "\"listrunning\"" = getRunningTaskList
100    | action == "\"listnew\"" = getNewTaskList
101    | action == "\"querystate\"" = getProgramState
102    | action == "\"clearprogram\"" = clearProgram
103    | otherwise = return $ Left (InvalidArgument $
104        "invalid action argument " ++ action)
105
106 -- 1. Find program name and verify stage.
107 -- 2. Execute program passing event and result.
108 -- 3. wait for result of program.
109 -- 4. Give result.
110 continueProgram :: (MonadIO m) => ActionCtxT ctx m (Either ServiceError String)
111 continueProgram = runEitherT $ do
112     prog <- lift (getArg "program") >>= hoistEither
113     stage <- lift (getArg "stage") >>= hoistEither
114     ident <- lift (getArg "id") >>= hoistEither
115     event <- lift (getArg "event") >>= hoistEither
116     result <- lift (getArg "result") >>= hoistEither
117     user <- lift (getArg "user") >>= hoistEither
118     platform <- lift (getArg "platform") >>= hoistEither
119     conn <- liftIO dbConnect >>= hoistEither
120     exec <- lift (getContinueExec conn prog ident user platform
121                     stage) >>= hoistEither
122     lift (removeWaitingProgram conn ident platform user prog) >>= hoistEither
123     liftIO (forkIO $ startContinueExec exec event result prog ident user platform)
124     liftIO (dbCommit conn) >>= hoistEither
125     liftIO (dbClose conn) >>= hoistEither
126     return "ok"
127
128 startContinueExec :: String -> String -> String -> String -> String ->
129     String -> String -> IO ()
130 startContinueExec exec event result prog ident startuser platform = do
131     putStrLn $ "server: start continue executable with result: " ++ show result
132     r <- catch (startContinueExec1 exec event result prog ident startuser platform) (
133         \e :: SomeException -> return . Left . InternalFailure $ show e)
134     case r of
135         Left e -> programFail e prog ident startuser platform
136         Right () -> return ()
137

```

```

138 failProgram :: (MonadIO m) => ActionCtxT ctx m (Either ServiceError String)
139 failProgram = runEitherT $ do
140     prog <- lift (getArg "program") >>= hoistEither
141     stage <- lift (getArg "stage") >>= hoistEither
142     ident <- lift (getArg "id") >>= hoistEither
143     msg <- lift (getArg "message") >>= hoistEither
144     user <- lift (getArg "user") >>= hoistEither
145     platform <- lift (getArg "platform") >>= hoistEither
146     conn <- liftIO dbConnect >>= hoistEither
147     lift (removeWaitingProgram conn ident platform user prog) >>= hoistEither
148     liftIO (programStrFail msg prog ident user platform)
149     liftIO (dbCommit conn) >>= hoistEither
150     liftIO (dbClose conn) >>= hoistEither
151     return "ok"
152
153 qesc :: String -> String
154 qesc = replace "'" "&SingQuote;"
155
156 startContinueExec1 :: String -> String -> String -> String -> String ->
157     String -> IO (Either ServiceError ())
158 startContinueExec1 exec event result prog ident startuser platform = runEitherT $ do
159     let cmd = exec++" "++qesc event++" "++qesc result++" "++qesc prog++" " ++
160         qesc ident++" "++qesc startuser++" "++qesc platform++" "
161     liftIO $ putStrLn ("server: continue command: "++cmd)
162     jsplat <- hoistEither (decodeJsonStr platform)
163     platStr <- hoistEither (jsonToStr jsplat)
164     let execPath = "/home/andreas/Dropbox/SDC/HsImpl1/Build/Output/"++platStr++"/"
165     liftIO $ putStrLn ("server: start executable " ++ prog)
166     (_, Just hout, _, hproc) <- liftIO $
167         createProcess (shell (execPath++cmd)) {
168             cwd = Just execPath,
169             env = Nothing,
170             std_in = Inherit,
171             std_out = CreatePipe,
172             std_err = Inherit,
173             close_fds = False,
174             create_group = False
175             --delegate_ctlc = False
176         }
177     ret <- liftIO $ waitForProcess hproc
178     hoistEither $ if ret == ExitSuccess
179         then Right ()
180         else Left (InternalFailure $ "failed executing subprogram: " ++ cmd)
181     output <- liftIO $ hGetContents hout
182     liftIO $ putStrLn ("server: decode continuation output " ++ output)
183     jout <- hoistEither $ decodeJsonStr output
184     liftIO $ putStrLn ("server: get JSON object")
185     jmap <- hoistEither $ jsonToMap jout
186     liftIO $ putStrLn ("server: get \"status\" from " ++ show jmap)
187     status <- hoistEither $ jsonObjLookup "status" jmap
188     liftIO $ putStrLn ("server: got \"status\" " ++ show status)
189     hoistEither $ if status /= strToJson "success"
190         then Left (InternalFailure output)
191         else Right ()
192     liftIO $ putStrLn ("server: success")

```

```

193     liftIO $ putStrLn ("server: " ++ output)
194     return ()
195
196 getContinueExec :: (MonadIO m) => Conn -> String -> String -> String ->
197     String -> String -> ActionCtxT ctx m (Either ServiceError String)
198 getContinueExec conn prog ident startuser startplat stage = runEitherT $ do
199     startplat1 <- hoistEither $ decodeJsonStr startplat
200     startuser1 <- hoistEither $ decodeJsonStr startuser
201     ident1 <- hoistEither $ decodeJsonStr ident
202     prog1 <- hoistEither $ decodeJsonStr prog
203     liftIO $ putStrLn ("server: get continue executable " ++ prog)
204     p <- liftIO $ dbGetProgressStage conn prog1 ident1 startuser1 startplat1
205     case p of
206         Left err -> hoistEither $
207             Left (InvalidArgument $ show err)
208         Right (i1,p1) -> do
209             i2 <- hoistEither $ maybeError (InvalidArgument
210                 "stage argument should be integer") (readMaybe stage)
211             liftIO $ putStrLn "server: got executable"
212             hoistEither $ if i1 /= i2
213                 then Left $ InvalidArgument ("unexpected value of stage argument")
214                 else Right p1
215
216 newProgram :: (MonadIO m) => ActionCtxT ctx m (Either ServiceError String)
217 newProgram = runEitherT $ do
218     prog <- lift (getArg "program") >>= hoistEither
219     ident <- lift (getArg "id") >>= hoistEither
220     user <- lift (getArg "user") >>= hoistEither
221     platform <- lift (getArg "platform") >>= hoistEither
222     conn <- liftIO dbConnect >>= hoistEither
223     exec <- lift (getNewExec conn prog platform) >>= hoistEither
224     liftIO (R.programStartup exec prog ident user platform) >>= hoistEither
225     liftIO (forkIO $ startNewExec exec prog ident user platform)
226     liftIO $ putStrLn ("server: reply ok")
227     liftIO (dbClose conn) >>= hoistEither
228     return "ok"
229
230 startNewExec :: String -> String -> String -> String -> IO ()
231 startNewExec exec prog ident user platform = do
232     r <- catch (startNewExec1 exec prog ident user platform) (
233         \e :: SomeException -> return . Left . InternalFailure $ show e)
234     case r of
235         Left e -> programFail e prog ident user platform
236         Right () -> return ()
237
238 programFail :: ServiceError -> String -> String -> String -> String -> IO ()
239 programFail err = programStrFail (show err)
240
241 programStrFail :: String -> String -> String -> String -> String -> IO ()
242 programStrFail err prog ident user platform = do
243     Right conn <- dbConnect
244     _ <- DB.run conn delW [DB.toSql ident, DB.toSql platform, DB.toSql user,
245         DB.toSql prog]
246     _ <- DB.run conn insF [DB.toSql ident, DB.toSql platform, DB.toSql user,
247         DB.toSql prog, DB.toSql err]

```

```

248     DB.commit conn
249     Right () <- liftIO $ dbClose conn
250     return ()
251     where delW = "DELETE FROM WaitingPrograms WHERE id = ? and startplatform = ? " ++
252           "and startuser = ? and program = ?"
253     insF = "INSERT INTO FailedPrograms VALUES (?,?,?,?,?,?)"
254
255 startNewExec1 :: String -> String -> String -> String -> String ->
256     IO (Either ServiceError ())
257 startNewExec1 exec prog ident user platform = runEitherT $ do
258     let cmd = exec++" "++qesc prog++" "++qesc ident++" "++qesc user++" "++qesc platform++"
259     "
260     jsplat <- hoistEither (decodeJsonStr platform)
261     platStr <- hoistEither (jsonToStr jsplat)
262     let execPath = "/home/andreas/Dropbox/SDC/HsImpl1/Build/Output/"++platStr++"/"
263     liftIO $ putStrLn ("server: startup command: "++execPath++cmd)
264     (_ , Just hout , _ , hproc) <- liftIO $ createProcess (shell (execPath++cmd)) {
265         cwd = Just execPath,
266         env = Nothing,
267         std_in = Inherit,
268         std_out = CreatePipe,
269         std_err = Inherit,
270         close_fds = False,
271         create_group = False
272         --delegate_ctlc = False
273     }
274     ret <- liftIO $ waitForProcess hproc
275     hoistEither $ if ret == ExitSuccess
276         then Right ()
277         else Left (InternalFailure $ "failed executing startup program: " ++ cmd)
278     output <- liftIO $ hGetContents hout
279     liftIO $ putStrLn ("server: decode startup output " ++ output)
280     jout <- hoistEither $ decodeJsonStr output
281     liftIO $ putStrLn ("server: get JSON object")
282     jmap <- hoistEither $ jsonToMap jout
283     liftIO $ putStrLn ("server: get \"status\"")
284     status <- hoistEither $ jsonObjLookup "status" jmap
285     liftIO $ putStrLn ("server: got \"status\" " ++ show status)
286     hoistEither $ if status /= strToJson "success"
287         then Left (InternalFailure output)
288         else Right ()
289     liftIO $ putStrLn ("server: success")
290     liftIO $ putStrLn ("server: " ++ output)
291     return ()
292
293 getNewExec :: (MonadIO m) => Conn -> String -> String ->
294     ActionCtxT ctx m (Either ServiceError String)
295 getNewExec conn prog plat = runEitherT $ do
296     eess <- liftIO (dbGetQuery conn query [DB.toSql prog, DB.toSql plat])
297     ess <- hoistEither eess
298     return $ DB.fromSql (ess!!0)
299     where query = "SELECT executable FROM Programs WHERE " ++
300           "program = ? and platform = ?"
301

```

```

302 removeWaitingProgram :: (MonadIO m) => Conn -> String -> String -> String ->
303     String -> ActionCtxT ctx m (Either ServiceError ())
304 removeWaitingProgram conn ident startplat startuser prog = runEitherT $ do
305     liftIO $ putStrLn "server: remove waiting program"
306     i <- liftIO $ DB.run conn delQ [
307         DB.toSql ident, DB.toSql startplat, DB.toSql startuser, DB.toSql prog]
308     hoistEither $ if i /= 1
309         then Right ()--Left (DoesNotExist "program not in waiting state")
310         else Right ()
311     where delQ = "DELETE FROM WaitingPrograms WHERE id = ? and startplatform = ? " ++
312             "and startuser = ? and program = ?"
313
314 addWaitingProgram :: (MonadIO m) => ActionCtxT ctx m (Either ServiceError String)
315 addWaitingProgram = runEitherT $ do
316     liftIO $ putStrLn "server: add waiting program"
317     ident <- lift (getArg "id") >>= hoistEither
318     plat <- lift (getArg "platform") >>= hoistEither
319     user <- lift (getArg "user") >>= hoistEither
320     prog <- lift (getArg "program") >>= hoistEither
321     startplat <- lift (getArg "startplatform") >>= hoistEither
322     startuser <- lift (getArg "startuser") >>= hoistEither
323     url <- lift (getArg "url") >>= hoistEither
324     conn <- liftIO dbConnect >>= hoistEither
325     exists1 <- lift $ userExists conn plat user
326     exists <- hoistEither exists1
327     if exists then do
328         liftIO $ putStrLn "server: user exists"
329         liftIO $ putStrLn "server: insert waiting program"
330         liftIO $ DB.run conn insQ [DB.toSql ident, DB.toSql plat, DB.toSql user,
331             DB.toSql prog, DB.toSql startplat, DB.toSql startuser, DB.toSql url]
332         liftIO $ DB.commit conn
333         liftIO $ putStrLn "server: added waiting program"
334         liftIO (dbClose conn) >>= hoistEither
335         return "ok"
336     else do
337         liftIO $ putStrLn "server: user does not exist"
338         hoistEither . Left $ DoesNotExist ("User "++user++" does not exist")
339         return "error"
340     where insQ = "INSERT INTO WaitingPrograms VALUES (?,?,?,?,?,?)"
341
342 userExists :: (MonadIO m) => Conn -> String -> String -> ActionCtxT ctx m (Either ServiceError
343     Bool)
344 userExists conn plat user = runEitherT $ do
345     liftIO $ putStrLn ("server: check that user "++user++" exists in platform "++plat)
346     res <- liftIO $ dbQuery conn query [DB.toSql user, DB.toSql plat]
347     res' <- hoistEither res
348     return $ if length res' > 0 then True else False
349     where query = "SELECT * FROM Users WHERE username=? and platform=?"
350
351 getNewTaskList :: (MonadIO m) => ActionCtxT ctx m (Either ServiceError String)
352 getNewTaskList = runEitherT $ do
353     liftIO $ putStrLn "server: get new task list"
354     conn <- liftIO dbConnect >>= hoistEither
355     group <- lift (getArg "group") >>= hoistEither
356     platform <- lift (getArg "platform") >>= hoistEither

```

```

356     rows <- liftIO $ DB.quickQuery conn query [
357         DB.toSql group, DB.toSql platform]
358     liftIO $ putStrLn "server: return new task list"
359     str <- liftIO $ mapM (\[p,e] -> return $
360         show (DB.fromSql p :: String, DB.fromSql e :: String)) rows
361     ret <- return $ "[" ++ intercalate "," str ++ "]"
362     liftIO (dbClose conn) >>= hoistEither
363     return ret
364     where query = "SELECT program,executable FROM Programs " ++
365             "WHERE usergroup <= ? and platform = ?"
366
367 getWaitingTaskList :: (MonadIO m) =>
368     ActionCtxT ctx m (Either ServiceError String)
369 getWaitingTaskList = runEitherT $ do
370     liftIO $ putStrLn "server: get waiting task list"
371     conn <- liftIO dbConnect >>= hoistEither
372     platform <- lift (getArg "platform") >>= hoistEither
373     username <- lift (getArg "user") >>= hoistEither
374     rows <- liftIO $ DB.quickQuery conn query [
375         DB.toSql platform, DB.toSql username]
376     liftIO $ putStrLn "server: return waiting task list"
377     str <- liftIO $ mapM (\[i,p,sp,su,u] -> return $ show (
378         DB.fromSql i :: String,
379         DB.fromSql p :: String,
380         DB.fromSql sp :: String,
381         DB.fromSql su :: String,
382         DB.fromSql u :: String)) rows
383     ret <- return $ "[" ++ intercalate "," str ++ "]"
384     liftIO (dbClose conn) >>= hoistEither
385     return ret
386     where query = "SELECT id,program,startplatform,startuser,url " ++
387             "FROM WaitingPrograms WHERE platform = ? and username = ?"
388
389 getFailedTaskList :: (MonadIO m) =>
390     ActionCtxT ctx m (Either ServiceError String)
391 getFailedTaskList = runEitherT $ do
392     liftIO $ putStrLn "server: get failed task list"
393     conn <- liftIO dbConnect >>= hoistEither
394     platform <- lift (getArg "platform") >>= hoistEither
395     username <- lift (getArg "user") >>= hoistEither
396     rows <- liftIO $ DB.quickQuery conn query [
397         DB.toSql platform, DB.toSql username]
398     str <- liftIO $ mapM (\[i,p,m] -> return $ show (
399         DB.fromSql i :: String,
400         DB.fromSql p :: String,
401         DB.fromSql m :: String)) rows
402     liftIO $ putStrLn "server: return failed task list"
403     ret <- return $ "[" ++ intercalate "," str ++ "]"
404     liftIO (dbClose conn) >>= hoistEither
405     return ret
406     where query = "SELECT id,program,message " ++
407             "FROM FailedPrograms WHERE platform = ? and username = ?"
408
409 getTerminatedTaskList :: (MonadIO m) =>
410     ActionCtxT ctx m (Either ServiceError String)

```

```

411 getTerminatedTaskList = runEitherT $ do
412     liftIO $ putStrLn "server: get terminated task list"
413     conn <- liftIO dbConnect >>= hoistEither
414     platform <- lift (getArg "platform") >>= hoistEither
415     username <- lift (getArg "user") >>= hoistEither
416     rows <- liftIO $ DB.quickQuery conn query [
417         DB.toSql platform, DB.toSql username]
418     str <- liftIO $ mapM (\[i,p] -> return $ show (
419             DB.fromSql i :: String,
420             DB.fromSql p :: String)) rows
421     liftIO $ putStrLn "server: return terminated task list"
422     ret <- return $ "[" ++ intercalate "," str ++ "]"
423     liftIO (dbClose conn) >>= hoistEither
424     return ret
425     where query = "SELECT id,program FROM TerminatedPrograms " ++
426                 "WHERE startplatform = ? and startuser = ?"
427
428 getRunningTaskList :: (MonadIO m) =>
429     ActionCtxT ctx m (Either ServiceError String)
430 getRunningTaskList = runEitherT $ do
431     liftIO $ putStrLn "server: get running task list"
432     conn <- liftIO dbConnect >>= hoistEither
433     platform <- lift (getArg "platform") >>= hoistEither
434     username <- lift (getArg "user") >>= hoistEither
435     rows <- liftIO $ DB.quickQuery conn query [
436         DB.toSql platform, DB.toSql username,
437         DB.toSql platform, DB.toSql username,
438         DB.toSql platform, DB.toSql username,
439         DB.toSql platform, DB.toSql username]
440     liftIO $ putStrLn "server: return running task list"
441     str <- liftIO $ mapM (\[i,_,_,p] -> return $ show (
442             DB.fromSql i :: String, DB.fromSql p :: String)) rows
443     ret <- return $ "[" ++ intercalate "," str ++ "]"
444     liftIO (dbClose conn) >>= hoistEither
445     return ret
446     where query = "SELECT id,startplatform,startuser,program FROM " ++
447                 "ProgramProgress WHERE startplatform = ? and startuser = ? " ++
448                 "EXCEPT SELECT id, platform as startplatform, " ++
449                 "username as startuser, program FROM WaitingPrograms WHERE " ++
450                 "startplatform = ? and startuser = ?" ++
451                 "EXCEPT SELECT id, platform as startplatform, " ++
452                 "username as startuser, program FROM FailedPrograms " ++
453                 "WHERE platform = ? and username = ?" ++
454                 "EXCEPT SELECT id, startplatform, startuser, program FROM " ++
455                 "TerminatedPrograms WHERE startplatform = ? and startuser = ?"
456
457 getProgramState :: (MonadIO m) => ActionCtxT ctx m (Either ServiceError String)
458 getProgramState = runEitherT $ do
459     liftIO $ putStrLn "server: get program state"
460     conn <- liftIO dbConnect >>= hoistEither
461     user <- lift (getArg "user") >>= hoistEither
462     plat <- lift (getArg "platform") >>= hoistEither
463     ident <- lift (getArg "id") >>= hoistEither
464     prog <- lift (getArg "program") >>= hoistEither
465     startuser <- lift (getArg "startuser") >>= hoistEither

```

```

466     startplat <- lift (getArg "startplatform") >>= hoistEither
467     liftIO $ putStrLn ("server: program state args are user:" ++ user ++
468                         ", plat:"++plat++, ident:"++ident++, prog:"++prog ++
469                         ", startplat:"++startplat++, startuser:"++startuser)
470     w <- liftIO $ dbGetQuery conn wQ [DB.toSql user, DB.toSql plat, DB.toSql ident,
471                                         DB.toSql prog, DB.toSql startuser, DB.toSql startplat]
472     ret <- case w of
473       Right w1 -> do
474         liftIO $ putStrLn "server: program state waiting"
475         return ("ProgramStateWaiting (" ++ DB.fromSql (w1!!0) ++ ")")
476       Left _ -> do
477         f <- liftIO $ dbGetQuery conn fQ [DB.toSql startuser,
478                                         DB.toSql startplat, DB.toSql ident, DB.toSql prog]
479         case f of
480           Right f1 -> do
481             liftIO $ putStrLn "server: program state failed"
482             return "ProgramStateFailed"
483           Left _ -> do
484             r <- liftIO $ dbGetQuery conn tQ [DB.toSql startuser,
485                                         DB.toSql startplat, DB.toSql ident, DB.toSql prog]
486             case r of
487               Right r1 -> do
488                 liftIO $ putStrLn "server: program state terminated"
489                 return "ProgramStateTerminated"
490               Left e -> do
491                 liftIO $ putStrLn ("server: then program is running: " ++ show e)
492                 return "ProgramStateRunning"
493     liftIO (dbClose conn) >>= hoistEither
494     return ret
495     where wQ = "SELECT url FROM WaitingPrograms WHERE username = ? " ++
496               "and platform = ? and id = ? and program = ? " ++
497               "and startuser = ? and startplatform = ?"
498     fQ = "SELECT message FROM FailedPrograms WHERE username = ? " ++
499          "and platform = ? and id = ? and program = ?"
500     tQ = "SELECT * FROM TerminatedPrograms WHERE startuser = ? " ++
501          "and startplatform = ? and id = ? and program = ?"
502
503 clearProgram :: (MonadIO m) => ActionCtxT ctx m (Either ServiceError String)
504 clearProgram = runEitherT $ do
505   liftIO $ putStrLn "server: start clear program"
506   conn <- liftIO dbConnect >>= hoistEither
507   user <- lift (getArg "user") >>= hoistEither
508   plat <- lift (getArg "platform") >>= hoistEither
509   ident <- lift (getArg "id") >>= hoistEither
510   prog <- lift (getArg "program") >>= hoistEither
511   liftIO . putStrLn $ "server: clear args user:"++user++, plat:"++plat ++
512                           ", ident:"++ident++, prog:"++prog
513   liftIO $ do
514     putStrLn "clear WaitingPrograms"
515     _ <- DB.run conn wU [DB.toSql ident, DB.toSql plat, DB.toSql user, DB.toSql prog]
516     putStrLn "clear ProgramProgress"
517     _ <- DB.run conn rU [DB.toSql ident, DB.toSql plat, DB.toSql user, DB.toSql prog]
518     putStrLn "clear FailedPrograms"
519     _ <- DB.run conn fU [DB.toSql ident, DB.toSql plat, DB.toSql user, DB.toSql prog]
520     putStrLn "clear TerminatedPrograms"

```

```

521      _ <- DB.run conn tU [DB.toSql ident, DB.toSql plat, DB.toSql user, DB.toSql prog]
522      putStrLn "clear State"
523      _ <- DB.run conn sU [DB.toSql ident, DB.toSql plat, DB.toSql user, DB.toSql prog]
524      return ()
525      liftIO (dbCommit conn) >>= hoistEither
526      liftIO (dbClose conn) >>= hoistEither
527      return "ok"
528      where wU = "DELETE FROM WaitingPrograms WHERE id = ? and startplatform = ? " ++
529            "and startuser = ? and program = ?"
530            rU = "DELETE FROM ProgramProgress WHERE id = ? and startplatform = ? " ++
531            "and startuser = ? and program = ?"
532            fU = "DELETE FROM FailedPrograms WHERE id = ? and platform = ? " ++
533            "and username = ? and program = ?"
534            tU = "DELETE FROM TerminatedPrograms WHERE id = ? and startplatform = ? " ++
535            "and startuser = ? and program = ?"
536            sU = "DELETE FROM State WHERE id = ? and startplatform = ? " ++
537            "and startuser = ? and program = ?"

```

LISTING D.6: Server.hs

```

1 {-# LANGUAGE OverloadedStrings, ScopedTypeVariables, DeriveDataTypeable, FlexibleContexts #-}
2
3 module PlatformInteractions (
4   makeInteraction,
5   mainServer,
6   fromEither,
7   fromMaybe,
8   homeButton
9 ) where
10
11 import Prelude as P
12 import qualified Data.Text as T
13 import Web.Spock.Safe
14 import Internal.Json
15 import Control.Monad.IO.Class (MonadIO, liftIO)
16 import ServiceError
17 import System.IO
18 import Internal.InteractionRuntime (InterArgs(..))
19 import qualified Data.ByteString.Char8 as BC
20 import qualified Network.Http.Client as C
21 import qualified System.IO.Streams.Internal as IOI
22 import Data.Typeable (Typeable)
23 import Control.Monad.CatchIO (catch)
24 import Control.Exception (throw,Exception,SomeException(..))
25 import Data.Aeson (encode)
26 import qualified Data.HashMap.Strict as Map
27
28 data PlatformException = PlatformException String
29   deriving (Typeable)
30
31 instance Exception PlatformException
32
33 instance Show PlatformException where
34   show (PlatformException s) = "PlatformException - "++s
35

```

```

36 addQuotes :: String -> String
37 addQuotes s = '\"':s++"\\""
38
39 removeQuotes :: String -> String
40 removeQuotes = filter ('"'/=)
41
42 makeInteraction :: (MonadIO m) => String -> ActionCtxT ctx m ()
43 makeInteraction platform = do
44     pid <- param "programId"
45     p <- param "program"
46     s <- param "programStage"
47     sp <- param "startPlatform"
48     su <- param "startUser"
49     iid <- param "interactionId"
50     a <- param "arguments"
51     let err = makeErrorInteraction platform pid p s sp su iid a
52     t <- liftIO (catch (doMakeInteraction platform pid p s sp su iid a) err)
53     if T.take 2 t == "ok"
54         then text t
55         else liftIO (err (PlatformException (T.unpack t))) >>= text
56
57 throwExcept :: String -> a
58 throwExcept = throw . PlatformException
59
60 makeErrorInteraction :: (MonadIO m, Exception e) => String -> Maybe String -> Maybe String ->
61     Maybe String -> Maybe String -> Maybe String -> e -> m (T.Text)
62 makeErrorInteraction platform pid p s sp su iid a ex = do
63     liftIO $ putStrLn ("platform: make error interaction for platform " ++ platform)
64     liftIO $ putStrLn ("platform: error is " ++ show ex)
65     let rec = InterArgs {programId=fromMaybe pid, program=fromMaybe p,
66                           programStage=fromMaybe s, startPlatform=fromMaybe sp,
67                           startUser=fromMaybe su, interactionId=fromMaybe iid,
68                           arguments=fromMaybe a}
69     let contArgs = [("action", "\\"continue\\"),
70                    ("id", BC.pack $ programId rec),
71                    ("program", BC.pack $ program rec),
72                    ("stage", BC.pack $ programStage rec),
73                    ("platform", BC.pack $ startPlatform rec),
74                    ("user", BC.pack $ startUser rec),
75                    ("event", "\\Error\\"),
76                    ("result", BC.pack $ "{\"Error\":\""++ show (encode (take 200 (show ex))) ++\"}")]
77     ] :: [(BC.ByteString,BC.ByteString)]
78     liftIO $ C.postForm mainServer contArgs $ \_ s -> do
79         bs <- IOI.read s
80         liftIO $ putStrLn ("platform: response " ++ show bs)
81         return $
82             case bs of
83                 Nothing -> "unexpected server response"
84                 Just x -> if not $ startsWithOk (BC.unpack x)
85                     then T.pack (BC.unpack x)
86                     else "ok"
87
88 doMakeInteraction :: (MonadIO m) => String -> Maybe String -> Maybe String -> Maybe String ->
89     Maybe String -> Maybe String -> Maybe String -> Maybe String -> m (T.Text)

```

```

90  doMakeInteraction platform pid p s sp su iid a = do
91      liftIO $ putStrLn ("platform: make interaction for platform " ++ platform)
92      let rec = InterArgs {programId=fromMaybe pid, program=fromMaybe p,
93                            programStage=fromMaybe s, startPlatform=fromMaybe sp,
94                            startUser=fromMaybe su, interactionId=fromMaybe iid,
95                            arguments=fromMaybe a}
96      let prefix1 = "http://localhost:5001/"++platform++"/show/"
97      let prefix2 = "/home/andreas/Dropbox/SDC/HsImpl1/Build/Output/"++platform++"/"
98      let urlPath = prefix1 ++ removeQuotes (program rec) ++
99                      removeQuotes (programId rec) ++
100                     removeQuotes (startUser rec) ++
101                     removeQuotes (startPlatform rec) ++ ".html"
102      let filePath = prefix2 ++ removeQuotes (program rec) ++
103                      removeQuotes (programId rec) ++
104                      removeQuotes (startUser rec) ++
105                      removeQuotes (startPlatform rec) ++ ".html"
106      liftIO $ putStrLn "platform: check interaction id"
107      identifyInteraction (interactionId rec) platform rec urlPath filePath
108
109 identifyInteraction :: (MonadIO m) => String -> String -> InterArgs -> String ->
110     String -> m (T.Text)
111 identifyInteraction name platform rec urlPath filePath
112     | name == "\"Message\"" =
113         makeInfoInteraction rec platform urlPath filePath
114     | name == "\"Table\"" =
115         makeTableInteraction rec platform urlPath filePath
116     | name == "\"Unary_input\"" =
117         makeOneInputInteraction rec platform urlPath filePath
118     | name == "\"Binary_input\"" =
119         makeTwoInputInteraction rec platform urlPath filePath
120     | name == "\"Binary_button\"" =
121         makeTwoButtonInteraction rec platform urlPath filePath
122     | otherwise =
123         throwError $ "unexpected interaction id " ++ interactionId rec
124
125 fromEither :: Either ServiceError a -> a
126 fromEither (Left e) = throwError (show e)
127 fromEither (Right x) = x
128
129 fromMaybe :: Maybe a -> a
130 fromMaybe Nothing = throwError "unable to fetch maybe value"
131 fromMaybe (Just x) = x
132
133 startsWithOk :: String -> Bool
134 startsWithOk [] = False
135 startsWithOk [_] = False
136 startsWithOk ('o': 'k': _) = True
137 startsWithOk _ = False
138
139 interactionPostForm :: (MonadIO m) => [(BC.ByteString, BC.ByteString)] -> m (T.Text)
140 interactionPostForm waitArgs =
141     liftIO $ C.postForm mainServer waitArgs $ \_ s -> do
142         bs <- IOI.read s
143         liftIO $ putStrLn ("platform: response " ++ show bs)
144         return $

```

```
145           case bs of
146             Nothing -> "unexpected server response"
147             Just x -> if not $ startsWithOk (BC.unpack x)
148               then T.pack (BC.unpack x)
149               else "ok"
150
151 interactionHead :: String -> String -> String
152 interactionHead plat user =
153   "<script src='/jquery.js'></script>" ++
154
155   "<script>" ++
156   "function runSubmit(form, submitForm, resultId, callback) {" ++
157     "var res = callback();" ++
158     "if (res == undefined) return false;" ++
159     "var f = document.getElementById(resultId).value = res;" ++
160
161     "$.ajax({" ++
162       "type: 'POST'," ++
163       "url: 'http://localhost:5005'," ++
164       "dataType: 'text'," ++
165       "crossDomain: true," ++
166       "data: $('#'+form).serialize()," ++
167       "success: function(msg) {" ++
168         "if (msg.startsWith('ok')) {" ++
169           "document.getElementById(submitForm).submit();" ++
170         "}else{" ++
171           "alert(msg);" ++
172         "}" ++
173       "}, " ++
174       "error: function(msg) {" ++
175         "alert('sorry, internal failure');" ++
176       "}" ++
177     ");" ++
178
179     "return false;" ++
180   "}" ++
181   "</script>" ++
182   homeButton plat user
183
184
185submitButton :: InterArgs -> String -> String -> String -> String -> String -> String
186submitButton rec user plat event resultCallback buttonText buttonId =
187  "<form id=\"_submitForm\"++buttonId++\" method=\"POST\" " ++
188    "action=\"/++removeQuotes plat++\"/running/program'>" ++
189    "<input type=\"hidden\" name=\"startuser\" value='++startUser rec++' />" ++
190    "<input type=\"hidden\" name=\"startplatform\" value='++startPlatform rec++' />" ++
191    "<input type=\"hidden\" name=\"program\" value='++program rec++' />" ++
192    "<input type=\"hidden\" name=\"user\" value='++user++' />" ++
193    "<input type=\"hidden\" name=\"id\" value='++programId rec++' />" ++
194  "</form>" ++
195
196  "<form id=\"_form\"++buttonId++\" method=\"POST\">" ++
197  "<input type=\"hidden\" name=\"action\" value='\"continue\"' />" ++
198  "<input type=\"hidden\" name=\"id\" value='++programId rec++' />" ++
199  "<input type=\"hidden\" name=\"program\" value='++program rec++' />" ++
```

```

200     "<input type=\"hidden\" name=\"stage\" value='++programStage rec++' />" ++
201     "<input type=\"hidden\" name=\"platform\" value='++startPlatform rec++' />" ++
202     "<input type=\"hidden\" name=\"user\" value='++startUser rec++' />" ++
203     "<input type=\"hidden\" name=\"event\" value='++event++' />" ++
204     "<input type=\"hidden\" name=\"result\" id='_formResult'++buttonId++' value=' />" ++
205     "<input value='++buttonText++' type=\"submit\" " ++
206     "onclick=\"return runSubmit(" ++
207         ", _form"++buttonId++", " ++
208         ", _submitForm"++buttonId++", " ++
209         ", _formResult"++buttonId++", " ++
210         resultCallback++) ;\\" >" ++
211   "</form>"
212
213 mainServer :: BC.ByteString
214 mainServer = "http://127.0.0.1:5005"
215
216 waitArgs :: InterArgs -> String -> String -> String -> [(BC.ByteString, BC.ByteString)]
217 waitArgs rec user platform url = [
218     ("action", "\"waiting\""),
219     ("id", BC.pack $ programId rec),
220     ("platform", BC.pack $ platform),
221     ("user", BC.pack $ user),
222     ("program", BC.pack $ program rec),
223     ("startplatform", BC.pack $ startPlatform rec),
224     ("startuser", BC.pack $ startUser rec),
225     ("url", BC.pack $ url)
226 ]
227
228 makeInfoInteraction :: (MonadIO m) => InterArgs -> String -> String -> String -> m (T.Text)
229 makeInfoInteraction rec platform urlPath filePath = do
230     liftIO $ putStrLn "platform: make \"Message\" interaction"
231     liftIO $ putStrLn ("platform: input args are " ++ arguments rec)
232     liftIO $ withFile filePath WriteMode (flip hPutStr htmlCode)
233     liftIO $ putStrLn "platform: tell main server interaction is ready"
234     interactionPostForm $
235         waitArgs rec userStr (addQuotes platform) (addQuotes urlPath)
236         where arg = fromEither $ decodeJsonStr (arguments rec)
237             aMap = fromEither $ jsonToMap arg
238             aStr1 = fromEither $ jsonObjLookup "\"Message\"" aMap
239             aUser = fromEither $ jsonObjLookup "\"User\"" aMap
240             aMsg = fromEither $ jsonToStr aStr1
241             userStr = encodeJsonStr aUser
242             htmlCode = "<body>" ++
243                 interactionHead platform userStr ++
244                 "<p>"++aMsg++"</p>" ++
245                 "<script>function callb(){return '{}'}</script>" ++
246                 submitButton rec userStr platform "\"Success\"" "callb" "OK" "OK" ++
247                 "</body>"
248
249 makeTableInteraction :: (MonadIO m) => InterArgs -> String -> String -> String -> m (T.Text)
250 makeTableInteraction rec platform urlPath filePath = do
251     liftIO $ putStrLn "platform: make \"Table\" interaction"
252     liftIO $ putStrLn ("platform: input args are " ++ arguments rec)
253     let table = fromEither (getInteractionTable aJsHd aList)
254     let htmlCode = "<body>" ++

```

```

255     interactionHead platform userStr ++
256     "<p>"++table++"</p>" ++
257     "<script>function callb(){return '{}';}</script>" ++
258     submitButton rec userStr platform "\"Success\" \"callb\" \"OK\" \"OK\" ++
259     "</body>""
260     liftIO $ withFile filePath WriteMode (flip hPutStr htmlCode)
261     liftIO $ putStrLn "platform: tell main server interaction is ready"
262     interactionPostForm $
263       waitArgs rec userStr (addQuotes platform) (addQuotes urlPath)
264     where arg = fromEither $ decodeJsonStr (arguments rec)
265       aMap = fromEither $ jsonToMap arg
266       aJsHd = fromEither $ jsonObjLookup "\"Header\"" aMap
267       aJs = fromEither $ jsonObjLookup "\"Table\"" aMap
268       aUser = fromEither $ jsonObjLookup "\"User\"" aMap
269       aList = fromEither $ jsonToList aJs
270       userStr = encodeJsonStr aUser
271
272     getInteractionTable :: Json -> [Json] -> Either ServiceError String
273     getInteractionTable hd js = do
274       hd' <- jsonToStr hd
275       ((("<table border='1'><tr><th>"++hd'++"</th></tr>")++) <$> (++"</table>") <$>
276       getInteractionTableFromStrings js
277     {-case getMaps js of
278       Right ms -> ("<table>"++) <$> (++"</table>") <$> getInteractionTableFromMaps ms
279       Left _ -> ("<table>"++) <$> (++"</table>") <$> getInteractionTableFromStrings js
280     where getMaps [] = return []
281       getMaps (j:js) = do
282         m <- jsonToMap j
283         ms <- getMaps js
284         return (m:ms)-}
285
286     {-getInteractionTableFromMaps :: [Map.HashMap T.Text Json] -> Either ServiceError String
287     getInteractionTableFromMaps [] = Right ""
288     getInteractionTableFromMaps (m:ms) = do
289       c <- contents
290       return $ header ++ c
291     where keys :: [String]
292       keys = map T.unpack $ Map.keys m
293       header = "<tr>" ++ getHeader keys ++ "</tr>"
294       getHeader [] = ""
295       getHeader (k:ks) = "<th>"++k++"</th>"++ getHeader ks
296       contents :: Either ServiceError String
297       contents = getContents (m:ms)
298       getContents :: [Map.HashMap T.Text Json] -> Either ServiceError String
299       getContents [] = Right ""
300       getContents (m:ms) = do
301         ds <- mapM (getData m) keys
302         rest <- getContents ms
303         return $ "<tr>"++concat ds++"</tr>"++rest
304     getData :: Map.HashMap T.Text Json -> String -> Either ServiceError String
305     getData m k = do
306       let js = fromMaybe (Map.lookup (T.pack k) m)
307       s <- jsonToStr js
308       return ("<td>"++s++"</td>")
309     -}

```

```

309
310 getInteractionTableFromStrings :: [Json] -> Either ServiceError String
311 getInteractionTableFromStrings ss = do
312     s <- doGet ss
313     return $ "<tr>"++s++"</tr>"
314     where doGet [] = Right ""
315         doGet (s:ss) = do
316             s' <- jsonToStr s
317             r <- getInteractionTableFromStrings ss
318             return ("<td>"++s'++"</td>"++r)
319
320 makeOneInputInteraction :: (MonadIO m) => InterArgs -> String -> String -> String -> m (T.Text)
321 makeOneInputInteraction rec platform urlPath filePath = do
322     liftIO $ putStrLn "platform: make \"Unary_input\" interaction"
323     liftIO $ putStrLn ("platform: input args are " ++ arguments rec)
324     liftIO $ withFile filePath WriteMode (flip hPutStr htmlCode)
325     liftIO $ putStrLn "platform: tell main server interaction is ready"
326     interactionPostForm $
327         waitArgs rec userStr (addQuotes platform) (addQuotes urlPath)
328     where arg = fromEither $ decodeJsonStr (arguments rec)
329         aMap = fromEither $ jsonToMap arg
330         aStr1 = fromEither $ jsonObjLookup "\"Message\"" aMap
331         aLbl = fromEither $ jsonObjLookup "\"Label\"" aMap
332         aLblStr = encodeJsonStr aLbl
333         aUser = fromEither $ jsonObjLookup "\"User\"" aMap
334         aMsg = fromEither $ jsonToStr aStr1
335         userStr = encodeJsonStr aUser
336         htmlCode = "<body>" ++
337             interactionHead platform userStr ++
338             "<p>"++aMsg++"</p>" ++
339             "<form>" ++
340             removeQuotes aLblStr ++ ":<br />" ++
341             "<input type='text' name='firstNumber' id='firstNumber' required></input><br />" ++
342             "</form>" ++
343             "<script>" ++
344             "function labelCallback() {" ++
345             "var f = document.getElementById('firstNumber').value;" ++
346             "return {'Input':f+''};" ++
347             "}" ++
348             "</script>" ++
349             submitButton rec userStr platform "\"Success\"" "labelCallback" "OK" "OK" ++
350             "</body>"
351
352 makeTwoInputInteraction :: (MonadIO m) => InterArgs -> String -> String -> String -> m (T.Text)
353 makeTwoInputInteraction rec platform urlPath filePath = do
354     liftIO $ putStrLn "platform: make \"Binary_input\" interaction"
355     liftIO $ putStrLn ("platform: input args are " ++ arguments rec)
356     liftIO $ withFile filePath WriteMode (flip hPutStr htmlCode)
357     liftIO $ putStrLn "platform: tell main server interaction is ready"
358     interactionPostForm $
359         waitArgs rec userStr (addQuotes platform) (addQuotes urlPath)
360     where arg = fromEither $ decodeJsonStr (arguments rec)
361         aMap = fromEither $ jsonToMap arg
362         aStr1 = fromEither $ jsonObjLookup "\"Message\"" aMap
363         aLblLeft = fromEither $ jsonObjLookup "\"Label1\"" aMap

```

```

364     aLblRight = fromEither $ jsonObjLookup "\"Label2\"" aMap
365     aLblLeftStr = encodeJsonStr aLblLeft
366     aLblRightStr = encodeJsonStr aLblRight
367     aUser = fromEither $ jsonObjLookup "\"User\"" aMap
368     aMsg = fromEither $ jsonToStr aStr1
369     userStr = encodeJsonStr aUser
370     htmlCode = "<body>" ++
371         interactionHead platform userStr ++
372         "<p>"++aMsg++"</p>" ++
373         "<form>" ++
374         removeQuotes aLblLeftStr ++ ":<br />" ++
375         "<input type='text' name='firstNumber' id='firstNumber' required></input><br />" ++
376         removeQuotes aLblRightStr ++ ":<br />" ++
377         "<input type='text' name='secondNumber' id='secondNumber' required></input>" ++
378         "</form>" ++
379         "<script>" ++
380         "function labelCallback() {" ++
381         "var f = document.getElementById('firstNumber').value;" ++
382         "var s = document.getElementById('secondNumber').value;" ++
383         "return '{\"Input1\":\"'"+f+"\", \"Input2\":\"'"+s+"\"}';" ++
384         "}" ++
385         "</script>" ++
386         submitButton rec userStr platform "\"Success\"" "labelCallback" "OK" "OK" ++
387         "</body>"
388
389 makeTwoButtonInteraction :: (MonadIO m) => InterArgs -> String -> String -> String -> m (T.Text)
390 makeTwoButtonInteraction rec platform urlPath filePath = do
391     liftIO $ putStrLn "platform: make \"Binary_button\" interaction"
392     liftIO $ putStrLn ("platform: input args are " ++ arguments rec)
393     liftIO $ withFile filePath WriteMode (flip hPutStr htmlCode)
394     liftIO $ putStrLn "platform: tell main server interaction is ready"
395     interactionPostForm $
396         waitArgs rec userStr (addQuotes platform) (addQuotes urlPath)
397         where arg = fromEither $ decodeJsonStr (arguments rec)
398             aMap = fromEither $ jsonToMap arg
399             aStr1 = fromEither $ jsonObjLookup "\"Message\"" aMap
400             aLblLeft = fromEither $ jsonObjLookup "\"Button_label1\"" aMap
401             aLblRight = fromEither $ jsonObjLookup "\"Button_label2\"" aMap
402             aLblLeftStr = encodeJsonStr aLblLeft
403             aLblRightStr = encodeJsonStr aLblRight
404             aUser = fromEither $ jsonObjLookup "\"User\"" aMap
405             aMsg = fromEither $ jsonToStr aStr1
406             userStr = encodeJsonStr aUser
407             htmlCode = "<body>" ++
408                 interactionHead platform userStr ++
409                 "<p>"++aMsg++"</p>" ++
410                 "<script>function callb(){return '{}'}</script>" ++
411                 submitButton rec userStr platform "\"Button1\"" "callb" (
412                     removeQuotes aLblLeftStr) "B1" ++
413                     submitButton rec userStr platform "\"Button2\"" "callb" (
414                         removeQuotes aLblRightStr) "B2" ++
415                     "</body>"
416
417 homeButton :: String -> String -> String
418 homeButton plat user =

```

```

419   "<button onclick=\"window.location.replace('' ++
420     "http://localhost:5001/"++removeQuotes plat++"/"++removeQuotes user++');" ++
421   "\">>Home</button>"

```

LISTING D.7: PlatformInteractions.hs

```

1 {-# LANGUAGE GADTs #-}
2 {-# LANGUAGE FlexibleInstances #-}
3
4 module Internal.ServiceIO (
5   ServiceIO(..),
6   SeqServiceIO,
7   runSeqServiceIO,
8   errorSIO,
9   fmapSIO,
10  applySIO,
11  returnSIO,
12  bindSIO,
13  liftSIO,
14  parSIO,
15  getVarSIO,
16  setVarSIO,
17  eitherSIO,
18  debugSIO
19 ) where
20
21 import ServiceError
22 import qualified Database.HDBC as DB
23 import Control.Concurrent.MVar (MVar,putMVar,readMVar,newEmptyMVar)
24 import Control.Concurrent (forkIO)
25 import Control.Applicative (Applicative(..))
26 import Control.Exception (try,catch)
27 import Data.Foldable (foldlM)
28 import qualified Data.Sequence as Seq
29 import Internal.Json
30 import Internal.Database (dbGetQuery,Conn,dbConnect)
31 import Internal.ProgramVariables
32 import System.IO (stderr,hPutStrLn)
33
34 {- Class for monads that supports invoking services -}
35 class (Functor m, Applicative m, Monad m) => ServiceIO m where
36   enterSIO :: m a -> SeqServiceIO a
37   leaveSIO :: SeqServiceIO a -> m a
38
39 {- The SeqServiceIO is running on top of the IO monad and is used to
40   - invoke services in sequence or in parallel by using the parSIO function. -}
41 data SeqServiceIO a = SeqServiceIO {
42   useState :: SeqServiceState a
43 }
44
45 type SeqServiceState a =
46   (AnyMVarSeq, Conn) -> IO (Either ServiceError a, (AnyMVarSeq, Conn))
47
48 data AnyMVar where
49   AnyMVar :: MVar (Either ServiceError a, AnyMVarSeq) -> AnyMVar

```

```
50
51 type AnyMVarSeq = Seq.Seq AnyMVar
52
53 instance ServiceIO SeqServiceIO where
54     enterSIO = id
55     leaveSIO = id
56
57 instance Monad SeqServiceIO where
58     return = returnSIO
59     (">>=) = bindSIO
60
61 instance Functor SeqServiceIO where
62     fmap = fmapSIO
63
64 instance Applicative SeqServiceIO where
65     pure = return
66     ( <*> ) = applySIO
67
68 leaveState :: (ServiceIO m) => SeqServiceState a -> m a
69 leaveState = leaveSIO . SeqServiceIO
70
71 returnSIO :: (ServiceIO m) => a -> m a
72 returnSIO x = leaveState $ \s -> return (Right x, s)
73
74 {- Indicate failure. Use this rather than fail! -}
75 errorSIO :: (ServiceIO m) => ServiceError -> m a
76 errorSIO e = leaveState $ \s -> return (Left e, s)
77
78 bindSIO :: (ServiceIO m) => m a -> (a -> m b) -> m b
79 bindSIO m f = leaveState $ \s -> do
80     (e, ns) <- useState (enterSIO m) s
81     case e of
82         Left e1 -> useState (errorSIO e1) ns
83         Right x -> catch (useState (enterSIO (f x)) ns) $ \e1 ->
84             useState (errorSIO $ ServiceException e1) ns
85
86 fmapSIO :: (Monad m) => (a -> b) -> m a -> m b
87 fmapSIO f m = m >>= return . f
88
89 applySIO :: (Functor m, Monad m) => m (a -> b) -> m a -> m b
90 applySIO f x = f >>= flip fmap x
91
92 liftSIO :: (ServiceIO m) => IO a -> m a
93 liftSIO m = leaveState $ \s -> (fmap (\x -> (Right x, s)) m)
94
95 eitherSIO :: (ServiceIO m) => Either ServiceError a -> m a
96 eitherSIO (Left e) = errorSIO e
97 eitherSIO (Right x) = return x
98
99 getConnSIO :: (ServiceIO m) => m Conn
100 getConnSIO = leaveState $ \(s,x) -> return (Right x, (s,x))
101
102 {- Store variable in database state -}
103 setVarSIO :: (ServiceIO m) => String -> Json -> m ()
104 setVarSIO i a = do
```

```

105    c <- getConnSIO
106    startuser <- liftSIO getStartUser
107    startplatform <- liftSIO getStartPlatform
108    ident <- liftSIO getId
109    prog <- liftSIO getProgram
110    _ <- liftSIO $ DB.run c delQ [DB.toSql prog, DB.toSql ident,
111                                DB.toSql startuser, DB.toSql startplatform, i1]
112    _ <- liftSIO $ DB.run c insQ [DB.toSql prog, DB.toSql ident,
113                                DB.toSql startuser, DB.toSql startplatform, i1, a1]
114    return ()
115    where a1 = DB.toSql $ encodeJson a
116        i1 = DB.toSql $ encodeJson (strToJson i)
117        delQ = "DELETE FROM State WHERE program = ? and id = ? " ++
118              "and startuser = ? and startplatform = ? and variable = ?"
119        insQ = "INSERT INTO State VALUES (?, ?, ?, ?, ?, ?)"
120
121 {- Store variable from database state -}
122 getVarSIO :: (ServiceIO m) => String -> m Json
123 getVarSIO i = do
124    c <- getConnSIO
125    startuser <- liftSIO getStartUser
126    startplatform <- liftSIO getStartPlatform
127    ident <- liftSIO getId
128    prog <- liftSIO getProgram
129    s1 <- liftSIO $ dbGetQuery c sQ [DB.toSql prog, DB.toSql ident,
130                                    DB.toSql startuser, DB.toSql startplatform, i1]
131    s <- either errorSIO return s1
132    either errorSIO return $ decodeJson (DB.fromSql $ head s)
133    where i1 = DB.toSql $ encodeJson (strToJson i)
134        sQ = "SELECT value FROM State WHERE program = ? and id = ? " ++
135              "and startuser = ? and startplatform = ? and variable = ?"
136
137 {-
138   - Spawn thread to concurrently execute first argument.
139   - Example which concurrently executes (return 0):
140   - test :: (ServiceIO m) => m Int
141   - test = do
142   -     c <- parSIO (return 0) -- start concurrent execution.
143   -     x <- c           -- wait for result of concurrent execution.
144   -     return x          -- return result of concurrent execution (0).
145 -}
146 parSIO :: (ServiceIO m) => m a -> m (m a)
147 parSIO m = leaveState $ \(s,c) -> do
148    v <- newEmptyMVar
149    _ <- forkIO $ do
150      (e,(x,_)) <- useState (enterSIO m) (Seq.empty,c)
151      putMVar v (e,x)
152      return (return . leaveState $ \(s1,_) -> do
153        (x,s2) <- readMVar v
154        return (x, (s1 Seq.>< s2, c)),
155        (s Seq.|> AnyMVar v, c))
156
157 {- Print debug message on server side. -}
158 debugSIO :: (ServiceIO m) => String -> m ()
159 debugSIO = liftSIO . hPutStrLn stderr

```

```

160
161 {- Start executing a sequence of service invocations. -}
162 runSeqServiceIO :: SeqServiceIO a -> IO (Either ServiceError a)
163 runSeqServiceIO m = do
164     conn <- dbConnect
165     case conn of
166         Left e -> return $ Left e
167         Right c -> runSeqServiceIO1 c m
168
169 runSeqServiceIO1 :: Conn -> SeqServiceIO a -> IO (Either ServiceError a)
170 runSeqServiceIO1 conn m = do
171     (r,(s,_)) <- useState m (Seq.empty,conn)
172     r1 <- foldlM pickFail r s
173     ret <- case r1 of
174         Left e -> DB.rollback conn >> return (Left e)
175         Right x -> DB.commit conn >> return (Right x)
176     DB.disconnect conn
177     return ret
178     where pickFail (Left e) _ = return (Left e)
179     pickFail r (AnyMVar v) = do
180         (e,s1) <- readMVar v
181         case e of
182             Left e1 -> return (Left e1)
183             -> foldlM pickFail r s1

```

LISTING D.8: Internal/ServiceIO.hs

```

1 module Internal.Json (
2     Json,
3     encodeJsonStr,
4     decodeJsonStr,
5     encodeJson,
6     decodeJson,
7     jsonToStr,
8     jsonToInt,
9     jsonToBool,
10    jsonToMap,
11    listToJson,
12    jsonToList,
13    jsonIsNull,
14    jsonObjLookup,
15    jsonToScientific,
16    strToJson,
17    intToJson,
18    boolToJson,
19    mapToJson,
20    scientificToJson,
21    isJsonMap
22 ) where
23
24 import ServiceError
25 import qualified Data.Aeson as A
26 import qualified Data.ByteString.Lazy as C
27 import qualified Data.ByteString.Lazy.Char8 as C8
28 import qualified Data.Text as T

```

```
29 import qualified Data.HashMap.Strict as Map
30 import qualified Data.Vector as Vec
31 import Data.Scientific
32
33 type Json = A.Value
34
35 type ObjectMap = Map.HashMap T.Text A.Value
36
37 decodeJsonStr :: String -> Either ServiceError Json
38 decodeJsonStr s = decodeJson (C8.pack s)
39
40 decodeJson :: C.ByteString -> Either ServiceError Json
41 decodeJson s =
42     case A.decode s :: Maybe A.Value of
43         Just json -> Right json
44         Nothing -> Left (JSONException $ "unable to parse json `" ++ C8.unpack s ++ `")"
45
46 encodeJsonStr :: Json -> String
47 encodeJsonStr = C8.unpack . encodeJson
48
49 encodeJson :: Json -> C.ByteString
50 encodeJson = A.encode
51
52 strToJson :: String -> Json
53 strToJson s = A.String (T.pack s)
54
55 jsonToStr :: Json -> Either ServiceError String
56 jsonToStr (A.String t) = Right (T.unpack t)
57 jsonToStr _ = Left (JSONException $ "invalid JSON string")
58
59 intToJson :: Integer -> Json
60 intToJson i = A.Number (fromInteger i)
61
62 boolToJson :: Bool -> Json
63 boolToJson = A.Bool
64
65 jsonToInt :: Json -> Either ServiceError Integer
66 jsonToInt (A.Number n) = Right (truncate n)
67 jsonToInt _ = Left (JSONException $ "invalid JSON number")
68
69 scientificToJson :: Scientific -> Json
70 scientificToJson = A.Number
71
72 jsonToScientific :: Json -> Either ServiceError Scientific
73 jsonToScientific (A.Number n) = Right n
74 jsonToScientific _ = Left (JSONException $ "invalid JSON number")
75
76 jsonToBool :: Json -> Either ServiceError Bool
77 jsonToBool (A.Bool b) = Right b
78 jsonToBool s = Left (JSONException $ "invalid JSON bool")
79
80 mapToJson :: ObjectMap -> Json
81 mapToJson = A.Object
82
83 jsonToMap :: Json -> Either ServiceError ObjectMap
```

```

84 jsonToMap (A.Object o) = Right o
85 jsonToMap s = Left (JSONException $ "invalid JSON object")
86
87 listToJson :: [A.Value] -> Json
88 listToJson = A.Array . Vec.fromList
89
90 jsonToList :: Json -> Either ServiceError [A.Value]
91 jsonToList (A.Array a) = Right (Vec.toList a)
92 jsonToList s = Left (JSONException $ "invalid JSON array")
93
94 isJsonMap :: Json -> Bool
95 isJsonMap (A.Object _) = True
96 isJsonMap _ = False
97
98 jsonIsNull :: Json -> Bool
99 jsonIsNull A.Null = True
100 jsonIsNull _ = False
101
102 jsonObjLookup :: String -> ObjectMap -> Either ServiceError A.Value
103 jsonObjLookup "" _ = Left (JSONException "empty JsonObject lookup string")
104 jsonObjLookup (c:cs) obj = let str = if c == '\'' then take (length cs - 1) cs else c:cs
105     in case Map.lookup (T.pack str) obj of
106         Nothing -> Left (JSONException $ "invalid JsonObject lookup " ++ (c:cs) ++ " in map " ++ show
107             obj)
107         Just x -> Right x

```

LISTING D.9: Internal/Json.hs

```

1 module Internal.ActIO (SeqActIO,runSeqActIO,ActIO) where
2
3 import ServiceError
4 import Control.Applicative (Applicative(..))
5 import Internal.ServiceIO
6 import qualified Internal.Services as In
7
8 class (ServiceIO m) => ActIO m
9
10 newtype SeqActIO a = SeqActIO (SeqServiceIO a)
11
12 instance ActIO SeqActIO
13
14 instance ServiceIO SeqActIO where
15     enterSIO (SeqActIO m) = m
16     leaveSIO = SeqActIO
17
18 instance Functor SeqActIO where
19     fmap = fmapSIO
20
21 instance Applicative SeqActIO where
22     pure = return
23     (<*>) = applySIO
24
25 instance Monad SeqActIO where
26     return = returnSIO
27     (=>) = bindSIO

```

```

28
29 {- Start a transaction (missing error/exception handling) -}
30 runSeqActIO :: (In.ActID -> SeqActIO a) -> IO (Either ServiceError a)
31 runSeqActIO act = do
32     aid <- In.startAct
33     e <- runSeqServiceIO . enterSIO $ act aid
34     either (\_ -> In.rollbackAct aid) (\_ -> In.commitAct aid) e
35     return e

```

LISTING D.10: Internal/ActIO.hs

```

1 {-# LANGUAGE FlexibleInstances, GADTs, OverloadedStrings, ScopedTypeVariables #-}
2
3 module Internal.InteractionRuntime where
4
5 import Prelude as P
6 import Data.Foldable (forM_)
7 import Control.Monad.IO.Class (liftIO)
8 import Control.Monad.Trans.Either
9 import Control.Monad (forM)
10 import Internal.ServiceIO
11 import Text.Read (read)
12 import qualified Data.Text as T
13 import Control.Exception as E
14 import qualified Data.HashMap.Strict as M
15 import Internal.Json
16 import ServiceError
17 import Internal.Database
18 import qualified Database.HDBC as DB
19 import System.Exit (exitWith, ExitCode (ExitSuccess))
20 import Network.Http.Client
21 import qualified Data.ByteString.Char8 as BC
22 import System.IO (stderr,hPutStrLn)
23 import qualified System.IO.Streams.Internal as IOI
24 import Control.Monad.Trans.Class (lift)
25 import Data.IORef
26 import Internal.ProgramVariables
27 import qualified Network.Http.Client as C
28 import qualified System.IO.Streams.Internal as IOI
29 import Data.Hashable (Hashable)
30 import Data.List (findIndex)
31 import System.Environment (getProgName)
32 import Data.String.Utils (replace)
33
34 data InterArgs = InterArgs {
35     programId :: String,
36     program :: String,
37     startPlatform :: String,
38     startUser :: String,
39     interactionId :: String,
40     programStage :: String,
41     arguments :: String
42 }
43
44 quote :: String -> String

```

```

45 quote = replace "&SingQuote;" ""
46
47 mainServer :: BC.ByteString
48 mainServer = "http://127.0.0.1:5005"
49
50 invert :: (Hashable k, Hashable v, Ord k, Ord v) => M.HashMap k v -> M.HashMap v k
51 invert m = M.fromList pairs
52     where pairs = [(v,k) | (k,v) <- M.toList m]
53
54 saveStateRecord :: String -> String -> String -> String -> SeqServiceIO ()
55 saveStateRecord mainprogram event s j =
56     if event /= "\\" Error \\
57     then normalSaveStateRecord s j
58     else errorSaveStateRecord mainprogram j
59
60 progToFuncName :: String -> String -> String
61 progToFuncName mainprogram s = let
62     Just si = findIndex (\c -> c == '_') s
63     s' = drop (si+1) s
64     in drop (length mainprogram - 1) s'
65
66 errorSaveStateRecord :: String -> String -> SeqServiceIO ()
67 errorSaveStateRecord mainprogram j = do
68     prog <- liftSIO $ getProgName
69     let func = progToFuncName mainprogram prog
70     json <- eitherSIO $ decodeJsonStr j
71     obj <- eitherSIO $ jsonToMap json
72     err <- eitherSIO $ jsonObjLookup "Error" obj
73     setVarSIO (func++"\_Error") err
74
75 normalSaveStateRecord :: String -> String -> SeqServiceIO ()
76 normalSaveStateRecord s j = do
77     vs <- liftSIO $ runEitherT saveState
78     case vs of
79         Left e -> errorSIO (CannotStartup (
80             "invalid interaction result " ++ show e))
81         Right vs1 -> forM_ vs1 $ \ (d,v) -> setVarSIO d v
82     where saveState = do
83         rec1 <- hoistEither $ decodeJsonStr s
84         rec <- hoistEither $ jsonToMap rec1
85         json <- hoistEither $ decodeJsonStr j
86         obj <- hoistEither $ jsonToMap json
87         forM (M.toList rec) $ 
88             \ (dest,src) -> do
89                 s1 <- hoistEither $ jsonToStr src
90                 v <- hoistEither $ jsonObjLookup s1 obj
91                 return (removeQuotes (T.unpack dest), v)
92
93 interactExec :: String -> String -> String -> String -> String ->
94     String -> String -> String -> IO (Either ServiceError ())
95 interactExec mainprogram ident user plat subprogram args evs name realPlat =
96     runEitherT $ do
97         conn <- liftIO dbConnect >>= hoistEither
98         r <- liftIO $ interactExec1 conn mainprogram ident user plat (
99             subprogram) args evs name realPlat

```

```

100      case r of
101          Left e -> liftIO $ DB.rollback conn
102          _ -> return ()
103      r1 <- hoistEither r
104      return r1
105
106 interactExec1 :: Conn -> String -> String -> String -> String -> String ->
107     String -> String -> String -> String -> IO (Either ServiceError ())
108 interactExec1 conn mainprogram ident user plat subprogram args evs name realPlat = runEitherT $ do
109     userJson <- hoistEither $ decodeJsonStr user
110     platJson <- hoistEither $ decodeJsonStr plat
111     identJson <- hoistEither $ decodeJsonStr ident
112     mainJson <- hoistEither $ decodeJsonStr mainprogram
113     (i,exec) <- liftIO (dbGetProgressStage conn mainJson identJson userJson $
114         platJson) >>= hoistEither
115     liftIO (dbSetProgressStage conn mainJson identJson userJson platJson (
116         intToJson (i+1)) subprogram) >>= hoistEither
117     liftIO (dbCommit conn) >>= hoistEither
118     let interArgs = InterArgs {
119         programId=ident,
120         program=mainprogram,
121         startPlatform=plat,
122         startUser=user,
123         interactionId=name,
124         programStage=show (i+1),
125         arguments=args
126     }
127     r <- liftIO (runInteract interArgs realPlat)
128     case r of
129         Right () -> liftIO . putStrLn $ "{\"status\":\"success\", \" ++
130             "\"next\":\""+++subprogram++"\", \"stage\":"+++show(i+1)++"}"
131         Left err -> do
132             r1 <- liftIO (dbSetProgressStage conn mainJson identJson userJson (
133                 platJson) (intToJson i) exec)
134             hoistEither r1
135             hoistEither (Left err)
136
137 readInteractArgs :: InterArgs -> IO (Either ServiceError InterArgs)
138 readInteractArgs args = runEitherT $ do
139     liftIO (runSeqServiceIO readIA) >>= hoistEither >>= hoistEither
140     where readIA = runEitherT $ do
141         a <- hoistEither $ decodeJsonStr (arguments args)
142         aM <- hoistEither $ jsonToMap a
143         res <- lift (flip mapM (M.toList aM) $ \ (k,v) -> do
144             val <- eitherSIO (jsonToStr v)
145             v1 <- getVarSIO val
146             return (k, v1))
147         return (args {arguments = encodeJsonStr (mapToJson (M.fromList res))})
148
149 runInteract :: InterArgs -> String -> IO (Either ServiceError ())
150 runInteract args plat = do
151     nArgs <- readInteractArgs args
152     case nArgs of
153         Left e -> return (Left e)
154         Right nArgs1 -> do

```

```

155         doRunInteract nArgs1 plat
156
157 removeQuotes :: String -> String
158 removeQuotes = P.filter ('"' /=-)
159
160 startsWithOk :: String -> Bool
161 startsWithOk [] = False
162 startsWithOk [_] = False
163 startsWithOk ('o': 'k': _) = True
164 startsWithOk _ = False
165
166 doRunInteract :: InterArgs -> String -> IO (Either ServiceError ())
167 doRunInteract args plat
168   | plat /= "\"Runtime\""
169     = do
170       let b = [("programId", BC.pack $ programId args),
171                 ("program", BC.pack $ program args),
172                 ("startPlatform", BC.pack $ startPlatform args),
173                 ("startUser", BC.pack $ startUser args),
174                 ("interactionId", BC.pack $ interactionId args),
175                 ("programStage", BC.pack $ programStage args),
176                 ("arguments", BC.pack $ arguments args)]
177       let url = BC.pack $ "http://127.0.0.1:5001/" ++
178             removeQuotes plat ++ "/make/interaction"
179       let runII = liftIO $ postForm url b $ \r s -> do
180         bs <- IOI.read s
181         return . Right $
182           case bs of
183             Nothing -> "Nothing"
184             Just x -> BC.unpack x
185         r <- E.catch runII $ \(e :: E.SomeException) -> do
186           return $ Left (InternalFailure $ show e)
187         return $ case r of
188           Left e -> Left e
189           Right s -> if s /= "ok"
190             then Left (InternalFailure $
191                         "unexpected interaction result: '" ++ s ++ "'")
192             else Right ()
193         | interactionId args == "\"if\""
194           = runEitherT $ do
195             js <- hoistEither $ decodeJsonStr (arguments args)
196             m <- hoistEither $ jsonToMap js
197             cnd <- hoistEither $ jsonObjLookup "cond" m
198             b <- hoistEither $ jsonToBool cnd
199             let event = if b then "\"True\"" else "\"False\""
200             let cArgs = [("action", "\"continue\""),
201                         ("id", BC.pack $ programId args),
202                         ("program", BC.pack $ program args),
203                         ("stage", BC.pack $ programStage args),
204                         ("platform", BC.pack $ startPlatform args),
205                         ("user", BC.pack $ startUser args),
206                         ("event", BC.pack event),
207                         ("result", "{}")]
208             resp <- liftIO $ C.postForm mainServer cArgs $ \_ s -> do
209               bs <- IOI.read s
210               case bs of
211                 Nothing -> return $ Left (InternalFailure "no server response")

```

```

210         Just bs1 -> if take 2 (BC.unpack bs1) /= "ok"
211             then return $ Left (InternalFailure $
212                 "unexpected server response " ++ show bs1)
213             else return $ Right ()
214
215             hoistEither resp
216
217 | interactionId args == "\"nop\""
218 = runEitherT $ do
219     let event = "\"Event\""
220     let cArgs = [("action", "\"continue\""),
221                 ("id", BC.pack $ programId args),
222                 ("program", BC.pack $ program args),
223                 ("stage", BC.pack $ programStage args),
224                 ("platform", BC.pack $ startPlatform args),
225                 ("user", BC.pack $ startUser args),
226                 ("event", BC.pack event),
227                 ("result", "{}")]
228
229     resp <- liftIO $ C.postForm mainServer cArgs $ \_ s -> do
230         bs <- IOI.read s
231         case bs of
232             Nothing -> return $ Left (InternalFailure "no server response")
233             Just bs1 -> if take 2 (BC.unpack bs1) /= "ok"
234                 then return $ Left (InternalFailure $
235                     "unexpected server response " ++ show bs1)
236                 else return $ Right ()
237
238             hoistEither resp
239
240 | interactionId args == "\"fail\""
241 = runEitherT $ do
242     js <- hoistEither $ decodeJsonStr (arguments args)
243     m <- hoistEither $ jsonToMap js
244     jmsg <- hoistEither $ jsonObjLookup "Error" m
245     msg <- hoistEither $ jsonToStr jmsg
246     let cArgs = [("action", "\"fail\""),
247                 ("id", BC.pack $ programId args),
248                 ("program", BC.pack $ program args),
249                 ("stage", BC.pack $ programStage args),
250                 ("platform", BC.pack $ startPlatform args),
251                 ("user", BC.pack $ startUser args),
252                 ("message", BC.pack msg)
253             ]
254
255     resp <- liftIO $ C.postForm mainServer cArgs $ \_ s -> do
256         bs <- IOI.read s
257         case bs of
258             Nothing -> return $ Left (InternalFailure "no server response")
259             Just bs1 -> if take 2 (BC.unpack bs1) /= "ok"
260                 then return $ Left (InternalFailure $
261                     "unexpected server response " ++ show bs1)
262                 else return $ Right ()
263
264             hoistEither resp
265
266 | otherwise = return (Left $ DoesNotExist (
267                 "no such Runtime interaction " ++ interactionId args))
268
269 anyStartup :: String -> String -> String -> String -> IO (Either ServiceError ())
270 anyStartup user plat prog ident = do
271     writeIORef ioRefStartUser user
272     writeIORef ioRefStartPlatform plat
273     writeIORef ioRefProgram prog
274     writeIORef ioRefId ident

```

```

265     let juser1 = decodeJsonStr user
266     case juser1 of
267         Left e -> return (Left e)
268         Right juser -> runSeqServiceIO $ setVarSIO "_username" juser
269
270 programStartup :: String -> String -> String -> String -> String ->
271     IO (Either ServiceError ())
272 programStartup exec p ident user plat = runEitherT $ do
273     userJson <- hoistEither $ decodeJsonStr user
274     platJson <- hoistEither $ decodeJsonStr plat
275     iJson <- hoistEither $ decodeJsonStr ident
276     pJson <- hoistEither $ decodeJsonStr p
277     conn <- liftIO dbConnect >>= hoistEither
278     curr <- liftIO (dbQuery conn query [DB.toSql p]) >>= hoistEither
279     hoistEither $
280         if length curr == 0
281             then Left (CannotStartup $ "unable to find program " ++ p)
282             else Right ()
283     old <- liftIO (dbGetProgressStage conn pJson iJson userJson platJson)
284     hoistEither $ case old of
285         Left _ -> Right ()
286         Right _ ->
287             Left (CannotStartup $ "program already exists," ++
288                   " try with another ID")
289     liftIO $ dbSetProgressStage conn pJson iJson userJson platJson (
290         intToJson 0) exec
291     liftIO (dbCommit conn) >>= hoistEither
292     where query = "SELECT program FROM Programs WHERE program = ?"
293
294 terminate :: String -> String -> String -> String -> String ->
295     IO (Either ServiceError ())
296 terminate mainprogram ident user plat exec = runEitherT $ do
297     userJson <- hoistEither $ decodeJsonStr user
298     platJson <- hoistEither $ decodeJsonStr plat
299     identJson <- hoistEither $ decodeJsonStr ident
300     mainJson <- hoistEither $ decodeJsonStr mainprogram
301     conn <- liftIO dbConnect >>= hoistEither
302     --liftIO (dbRemoveProgressStage conn mainJson identJson userJson platJson) >>=
303     --    hoistEither
304     liftIO $ DB.run conn waitQ [DB.toSql ident, DB.toSql mainprogram,
305         DB.toSql plat, DB.toSql user]
306     liftIO $ DB.run conn insQ [DB.toSql mainprogram, DB.toSql ident,
307         DB.toSql user, DB.toSql plat]
308     liftIO (dbCommit conn) >>= hoistEither
309     liftIO . putStrLn $ "{\"status\":\"success\", \" ++
310         \"next\":\\\"\\\", \"stage\":\"++show(-1)++\"}"
311     liftIO $ exitWith ExitSuccess
312     where waitQ = "DELETE FROM WaitingPrograms WHERE " ++
313             "id = ? and program = ? and startplatform = ? and startuser = ?"
314     insQ = "INSERT INTO TerminatedPrograms VALUES (?, ?, ?, ?, ?)"

```

LISTING D.11: Internal/InteractionRuntime.hs

```

1 module Internal.TestSeqServiceIO where
2

```

```
3 import SeqServiceIO
4 import Services
5 import Data.Foldable (foldlM)
6
7 testSeqServiceIO :: (Show a) => SeqServiceIO a -> IO ()
8 testSeqServiceIO act = do
9     r <- runSeqServiceIO act
10    case r of
11        Left e -> putStrLn $ "error: " ++ show e
12        Right x -> putStrLn $ "success: " ++ show x
13
14 test1 :: SeqServiceIO String
15 test1 = do
16     x <- serviceSuccess
17     y <- serviceSuccess
18     return (x ++ " and " ++ y)
19
20 test2 :: SeqServiceIO String
21 test2 = do
22     x <- serviceSuccess
23     y <- serviceDoesNotExistError
24     z <- serviceSuccess
25     return (x ++ " and " ++ y ++ " and " ++ z)
26
27 test3 :: SeqServiceIO String
28 test3 = do
29     x <- serviceSuccess
30     y <- serviceException
31     z <- serviceSuccess
32     return (x ++ " and " ++ y ++ " and " ++ z)
33
34 test4 :: SeqServiceIO String
35 test4 = do
36     x <- parSIO serviceSuccess
37     y <- parSIO serviceSuccess
38     z <- parSIO serviceSuccess
39     x1 <- x
40     y1 <- y
41     z1 <- z
42     return (x1 ++ " and " ++ y1 ++ " and " ++ z1)
43
44 test5 :: SeqServiceIO String
45 test5 = do
46     x <- parSIO serviceSuccess
47     y <- parSIO serviceException
48     z <- parSIO serviceSuccess
49     x1 <- x
50     y1 <- y
51     z1 <- z
52     return (x1 ++ " and " ++ y1 ++ " and " ++ z1)
53
54 test6 :: SeqServiceIO String
55 test6 = do
56     let iters = 10000
57     xs <- mapM (const $ parSIO service1) (replicate iters ())
```

```

58     s <- foldLM (\s x -> do
59         x1 <- x
60         return (s+x1)) 0 xs
61     return $ " ? " ++ show (iters == fromInteger s)

```

LISTING D.12: Internal/TestSeqServiceIO.hs

```

1 module Internal.Database (
2     dbGetQuery,
3     dbGetProgressStage,
4     dbRemoveProgressStage,
5     dbSetProgressStage,
6     Conn,
7     dbConnect,
8     dbCommit,
9     dbQuery,
10    dbClose
11 ) where
12
13 import ServiceError
14 import Control.Monad.IO.Class (liftIO)
15 import Control.Monad.Trans.Either
16 import Control.Exception (catch)
17 import qualified Database.HDBC.PostgreSQL as P
18 import qualified Database.HDBC as DB
19 import Internal.Json
20
21 {- Sqlite3 database connection type -}
22 type Conn = P.Connection
23
24 dbGetQuery :: Conn -> String -> [DB.SqlValue] ->
25     IO (Either ServiceError [DB.SqlValue])
26 dbGetQuery conn query args = runEitherT $ do
27     s <- liftIO (dbQuery conn query args) >>= hoistEither
28     r <- hoistEither $ if length s /= 1
29         then Left (DoesNotExist ("unable to find database row " ++
30             query ++" ["++ show args ++"]"))
31         else Right (head s)
32     return r
33
34 dbQuery :: Conn -> String -> [DB.SqlValue] ->
35     IO (Either ServiceError [[DB.SqlValue]])
36 dbQuery conn query args = catch (fmap Right (DB.quickQuery conn query args)) $
37     return . Left . ServiceException
38
39 dbConnect :: IO (Either ServiceError Conn)
40 dbConnect = catch (fmap Right (P.connectPostgreSQL connectStr)) $
41     return . Left . ServiceException
42     where connectStr = "host=localhost dbname=sdc user=sdc password=sdc"
43
44 dbClose :: Conn -> IO (Either ServiceError ())
45 dbClose conn = do
46     c <- liftIO (dbCommit conn)
47     case c of
48         Left e -> return (Left e)

```

```

49     Right () ->
50         catch (fmap Right (DB.disconnect conn)) $ return . Left . ServiceException
51
52 dbCommit :: Conn -> IO (Either ServiceError ())
53 dbCommit conn = liftIO $ catch (fmap Right (DB.commit conn)) (
54     return . Left . ServiceException)
55
56 dbGetProgressStage :: Conn -> Json -> Json -> Json -> Json ->
57     IO (Either ServiceError (Integer, String))
58 dbGetProgressStage conn p ident user plat = runEitherT $ do
59     let i2 = encodeJsonStr ident
60     let p2 = encodeJsonStr p
61     let user2 = encodeJsonStr user
62     let plat2 = encodeJsonStr plat
63     r <- liftIO $ dbGetQuery conn ("SELECT next_exec,stage " ++
64         "FROM ProgramProgress WHERE program = ? and id = ? and " ++
65         "startuser = ? and startplatform = ?") [
66             DB.toSql p2, DB.toSql i2, DB.toSql user2, DB.toSql plat2]
67     r1 <- hoistEither r
68     return (DB.fromSql (r1 !! 1), DB.fromSql (r1 !! 0))
69
70 dbRemoveProgressStage :: Conn -> Json -> Json -> Json -> Json ->
71     IO (Either ServiceError ())
72 dbRemoveProgressStage conn prog ident user plat = runEitherT $ do
73     let i2 = encodeJsonStr ident
74     let p2 = encodeJsonStr prog
75     let user2 = encodeJsonStr user
76     let plat2 = encodeJsonStr plat
77     _ <- liftIO $ DB.run conn (
78         "DELETE FROM ProgramProgress WHERE " ++
79         "program = ? and id = ? and startuser = ? and startplatform = ?") [
80             DB.toSql p2, DB.toSql i2, DB.toSql user2, DB.toSql plat2]
81     return ()
82
83 dbSetProgressStage :: Conn -> Json -> Json -> Json -> Json -> String ->
84     IO (Either ServiceError ())
85 dbSetProgressStage conn p ident user plat s exec = runEitherT $ do
86     let i2 = encodeJsonStr ident
87     let p2 = encodeJsonStr p
88     let user2 = encodeJsonStr user
89     let plat2 = encodeJsonStr plat
90     let s2 = encodeJsonStr s
91     liftIO (dbRemoveProgressStage conn p ident user plat) >>= hoistEither
92     _ <- liftIO $ DB.run conn (
93         "INSERT INTO ProgramProgress VALUES (?,?,?,?,?,?)") [DB.toSql p2,
94             DB.toSql i2, DB.toSql plat2, DB.toSql user2, DB.toSql exec, DB.toSql s2]
95     r <- liftIO $ dbCommit conn
96     hoistEither r

```

LISTING D.13: Internal/Database.hs

```

1 module Internal.ProgramVariables where
2
3 import Data.IORef
4 import Data.Global

```

```

5   ioRefStartUser :: IORef String
6   ioRefStartUser = declareIORef "GlobalStartUser" ""
7
8   ioRefStartPlatform :: IORef String
9   ioRefStartPlatform = declareIORef "GlobalStartPlatform" ""
10
11  ioRefProgram :: IORef String
12  ioRefProgram = declareIORef "GlobalProgram" ""
13
14  ioRefId :: IORef String
15  ioRefId = declareIORef "GlobalId" ""
16
17
18  getStartUser :: IO String
19  getStartUser = readIORef ioRefStartUser
20
21  getStartPlatform :: IO String
22  getStartPlatform = readIORef ioRefStartPlatform
23
24  getProgram :: IO String
25  getProgram = readIORef ioRefProgram
26
27  getId :: IO String
28  getId = readIORef ioRefId

```

LISTING D.14: Internal/ProgramVariables.hs

```

1 module Internal.TestSeqActIO where
2
3 import Data.Foldable (foldlM)
4
5 import SeqActIO
6 import Services
7 import SeqServiceIO
8
9 testSeqActIO :: (Show a) => (ActID -> SeqActIO a) -> IO ()
10 testSeqActIO act = do
11     r <- runSeqActIO act
12     case r of
13         Left e -> putStrLn $ "error: " ++ show e
14         Right x -> putStrLn $ "success: " ++ show x
15
16 test1 :: ActID -> SeqActIO String
17 test1 _ = do
18     x <- serviceSuccess
19     y <- serviceSuccess
20     return (x ++ " and " ++ y)
21
22 test2 :: ActID -> SeqActIO String
23 test2 _ = do
24     x <- serviceSuccess
25     y <- serviceDoesNotExistError
26     z <- serviceSuccess
27     return (x ++ " and " ++ y ++ " and " ++ z)
28

```

```

29 test3 :: ActID -> SeqActIO String
30 test3 _ = do
31     x <- serviceSuccess
32     y <- serviceException
33     z <- serviceSuccess
34     return (x ++ " and " ++ y ++ " and " ++ z)
35
36 test4 :: ActID -> SeqActIO String
37 test4 _ = do
38     x <- parSIO serviceSuccess
39     y <- parSIO serviceSuccess
40     z <- parSIO serviceSuccess
41     x1 <- x
42     y1 <- y
43     z1 <- z
44     return (x1 ++ " and " ++ y1 ++ " and " ++ z1)
45
46 test5 :: ActID -> SeqActIO String
47 test5 _ = do
48     _ <- parSIO serviceSuccess
49     _ <- parSIO serviceException
50     _ <- parSIO serviceSuccess
51     return "this should not get printed"
52
53 test6 :: ActID -> SeqActIO String
54 test6 _ = do
55     let iters = 100000
56     xs <- mapM (const $ parSIO service1) (replicate iters ())
57     s <- foldlM (\s x -> do
58         x1 <- x
59         return (s+x1)) 0 xs
60     return $ " ? " ++ show (iters == fromInteger s)
61
62 test7 :: ActID -> SeqActIO String
63 test7 aid = do
64     x <- parSIO (test4 aid)
65     y <- parSIO (test4 aid)
66     x1 <- x
67     y1 <- y
68     return (x1 ++ " and " ++ y1)
69
70 test8 :: ActID -> SeqActIO String
71 test8 aid = do
72     _ <- parSIO (test4 aid)
73     _ <- parSIO (test5 aid)
74     return "this should not get printed"
75
76 test9 :: ActID -> SeqActIO ()
77 test9 _ = actService

```

LISTING D.15: Internal/TestSeqActIO.hs

```

1 module Internal.Services (
2     ActID(..),
3     startAct,

```

```

4     commitAct,
5     rollbackAct) where
6
7 newtype ActID = ActID Int deriving (Eq,Show)
8
9 startAct :: IO ActID
10 startAct = return (ActID 0)
11
12 commitAct :: ActID -> IO ()
13 commitAct _ = return ()
14
15 rollbackAct :: ActID -> IO ()
16 rollbackAct _ = return ()

```

LISTING D.16: Internal/Services.hs

```

1 {-# LANGUAGE NoMonomorphismRestriction #-}
2
3 module Internal.Json (
4     Json,
5     encodeJson,
6     decodeJson,
7     jsonToStr,
8     jsonToInt,
9     json.ToDouble,
10    jsonToBool,
11    jsonToMap,
12    jsonToList,
13    jsonIsNull,
14    jsonObjLookup,
15    strToJson,
16    intToJson,
17    mapToJson
18 ) where
19
20 import Data.List (foldl', intersperse)
21 import Data.Char (chr)
22 import Data.List (intercalate)
23 import qualified Data.Map as Map
24 import ServiceError
25 import Debug.Trace
26
27 import Text.Parsec
28
29 data Json = JsonNum Double
30     | JsonStr String
31     | JsonArr [Json]
32     | JsonObj (Map.Map Json Json)
33     | JsonBool Bool
34     | JsonNull
35     deriving (Eq, Ord)
36
37 -- Collapse a Json document to a string
38 -- If this confuses you, just accept it as magic.
39 instance Show Json where

```

```

40     showsPrec _ = showsJSON
41     where commaJoin = foldl' (.) id . intersperse (", "++)
42         showsAssoc (field, val) = shows field . (" :"++) . shows val
43         showsJSON (JsonNum num) = shows num
44         showsJSON (JsonStr str) = shows str
45         showsJSON (JsonArr jarr) =
46             ('[':) . commaJoin (map shows jarr) . (']':)
47         showsJSON (JsonObj aa) =
48             ('{':) . commaJoin (map showsAssoc $ Map.toList aa) . ('}':)
49         showsJSON (JsonBool tf) =
50             if tf then ("true"++) else ("false"++)
51         showsJSON JsonNull      = ("null"++)
52
53 -- The base parser
54 jsonParser = do
55     spaces -- handle whitespace
56     -- Each of these parses the corresponding data type
57     -- E.g. jsonPStr -> JsonStr
58     json <- jsonPNum
59         <|> jsonPStr
60         <|> jsonPBool
61         <|> jsonPNull
62         <|> jsonPArr
63         <|> jsonPAA
64         <?> "value" -- Otherwise report we expected a value
65     spaces
66     -- return for monads means something very different than
67     -- + procedural languages.
68     -- Simply put, it wraps the value in the monad
69     return json
70
71 -- These parsers try to collect a literal as a string.
72 -- Note that the try is not strictly necessary, but this is a good example
73 -- + of when one might use it (Consider if there was also a "none" value)
74 jsonPNull = try (string "null") >> return JsonNull
75
76 jsonPBool = fmap JsonBool
77     $ (try (string "true") >> return True)
78     <|> (try (string "false") >> return False)
79
80 -- Note how this parser splits the problem into parts
81 jsonPNum = do
82     -- We return a function that might negate our result later
83     doNeg <- (char '-' >> return negate) <|> return id
84     -- Every number can
85     realPart <- realP
86     fracPart <- fracP
87     expPart <- expP
88     -- We put all the pieces together and wrap it into a Json type
89     return $ JsonNum $ doNeg $ (realPart + fracPart) * 10 ** expPart
90     -- This is the real part. It must exist.
91     where realP = (char '0' >> return 0) <|> (do
92             -- It's either just a 0, or begins with 1 - 9
93             firstDigit <- satisfy (\ ch -> '1' <= ch && ch <= '9')
94             -- after that it can be any digit

```

```

95         restDigits <- many digit
96         return $ read $ firstDigit : restDigits
97     )
98     fracP = (do
99         char '.'
100        digits <- many1 digit
101        return $ read $ "0." ++ digits
102    )
103        -- This parser will fail if there's no dot
104        -- That means there's no fractional part, so we default to 0
105        <|> return 0
106     expP = (do
107         oneOf "eE"
108         -- This is similar to the other doNeg, but we have an
109         -- + extra option
110         doNeg <- (char '-' >> return negate)
111         <|> (char '+' >> return id)
112         <|> return id
113         digits <- many1 digit
114         return $ doNeg $ read digits
115     )
116         -- Similarly, we default to 0 if there's no exponent
117     <|> return 0
118
119 jsonStr = do
120     char '\"'
121     s <- strChar `manyTill` (char '\"')
122     return s
123     -- This is a list of substitutions to make after a \
124 where subMap = Map.fromList
125     [ ('\"', '\"')
126     , ('\\\", '\"')
127     , ('/', '/')
128     , ('n', '\n')
129     , ('r', '\r')
130     , ('f', '\f')
131     , ('t', '\t')
132     , ('b', '\b')
133     ]
134     -- Parses a
135     unicode = do
136         -- get 4 hexadecimal digits
137         code <- count 4 hexDigit
138         if null code -- The docs say this is possible
139             then fail $ "expecting unicode"
140             -- Otherwise, just read the values
141             else return $ chr $ read $ "0x" ++ code
142         -- This acts like aanyChar, but will try to escape first
143     strChar = (do
144         char '\\'
145         ch <- anyChar
146         -- search for the correct substitution
147         case Map.lookup ch subMap of
148             Just newCh -> return newCh
149             Nothing -> do

```

```

150             if ch == 'u' -- It might yet be unicode
151                 then unicode
152                 else fail "expecting escape sequence"
153             )
154             -- If we don't have an escape sequence, just return the char
155             <|> anyChar
156
157             -- This just wraps the String result of jsonStr into a Json
158             jsonPStr = fmap JsonStr jsonStr
159
160             -- Arrays are real simple at this point.
161             -- Just get a comma seperated list of Json values between brackets
162             jsonPArr = do
163                 char '['
164                 jarr <- jsonParser `sepBy` (char ',')
165                 char ']'
166                 return $ JsonArr $ jarr
167
168             -- Associative arrays are almost as simple, except we need to teach it
169             -- + how to parse fields
170             jsonPAA = do
171                 char '{'
172                 jAA <- assocP `sepBy` char ',' -- returns a list of (key, value) tuples
173                 char '}'
174                 return $ JsonObject $ Map.fromList $ jAA
175                 -- This should seem really straight forward.
176                 -- We get the label followed by a colon and its corresponding
177                 -- + Json value
178             where assocP = do
179                 spaces
180                 label <- jsonParser
181                 spaces
182                 char ':'
183                 json <- jsonParser
184                 return (label, json)
185
186             decodeJson :: String -> Either ServiceError Json
187             decodeJson s =
188                 case parse jsonParser "" s of
189                     Right json -> Right json
190                     Left err -> Left (JSONException (show err))
191
192             encodeJson :: Json -> String
193             encodeJson (JsonStr s) = '"' : s ++ "\""
194             encodeJson (JsonNum n) = show n
195             encodeJson (JsonBool True) = "true"
196             encodeJson (JsonBool False) = "false"
197             encodeJson JsonNull = "null"
198             encodeJson (JsonObject o) = "{" ++ pairs (Map.toList o) ++ "}"
199             where pairs [] = ""
200                 pairs ps = intercalate ", " (map renderPair ps)
201                 renderPair (k,v) = show k ++ ":" ++ encodeJson v
202             encodeJson (JsonArr a) = "[" ++ values a ++ "]"
203             where values [] = ""
204                 values vs = intercalate ", " (map encodeJson vs)

```

```

205
206 strToJson s = JsonStr s
207
208 jsonToStr (JsonStr s) = Right (':':s++"\n")
209 jsonToStr s = Left (JSONException $ "invalid JsonStr" ++ show s)
210
211 intToJson i = JsonNum (fromInteger i)
212
213 jsonToInt :: Json -> Either ServiceError Integer
214 jsonToInt (JsonNum n) = Right (truncate n)
215 jsonToInt s = Left (JSONException $ "invalid JsonNum" ++ show s)
216
217 json.ToDouble (JsonNum n) = Right n
218 json.ToDouble s = Left (JSONException $ "invalid JsonNum" ++ show s)
219
220 jsonToBool (JsonBool b) = Right b
221 jsonToBool s = Left (JSONException $ "invalid JsonBool" ++ show s)
222
223 mapToJson = encodeJson . JsonObj
224
225 jsonToMap (JsonObj o) = Right o
226 jsonToMap s = Left (JSONException $ "invalid JsonObj" ++ show s)
227
228 jsonToList (JsonArr a) = Right a
229 jsonToList s = Left (JSONException $ "invalid JsonArr" ++ show s)
230
231 jsonIsNull v = v == JsonNull
232
233 jsonObjLookup "" _ = Left (JSONException "empty JsonObj lookup string")
234 jsonObjLookup (c:cs) obj = let str = if c == '\'' then take (length cs - 1) cs else c:cs
235     in case Map.lookup (strToJson str) obj of
236         Nothing -> Left (JSONException $ "invalid JsonObj lookup" ++ (c:cs) ++ " in map" ++ show
237             obj)
238         Just x -> Right x

```

LISTING D.17: Internal/OldJson.hs

```

1 module Services (
2     ActID,
3     serviceSuccess,
4     serviceDoesNotExistError,
5     serviceException,
6     service1,
7     actService,
8     invokePostService) where
9
10 import SeqServiceIO
11 import Internal.ServiceIO (liftSIO)
12 import Internal.ActIO (ActIO)
13 import Internal.Services (ActID)
14 import ServiceError
15 import qualified Network.Http.Client as C
16 import qualified Data.ByteString.Char8 as BC
17 import qualified System.IO.Streams.Internal as IOI
18

```

```

19 serviceSuccess :: (ServiceIO m) => m String
20 serviceSuccess = do
21     liftSIO $ putStrLn "service success"
22     return "success"
23
24 serviceDoesNotExistError :: (ServiceIO m) => m String
25 serviceDoesNotExistError = do
26     liftSIO $ putStrLn "service does not exist error"
27     errorSIO (DoesNotExist "error message")
28
29 serviceException :: (ServiceIO m) => m String
30 serviceException = do
31     liftSIO $ putStrLn "service exception"
32     liftSIO $ ioError (userError "IO exception")
33
34 service1 :: (ServiceIO m) => m Integer
35 service1 = return 1
36
37 actService :: (ActIO m) => m ()
38 actService = do
39     liftSIO $ putStrLn "act service"
40
41 invokePostService :: (ServiceIO m) => String -> [(String, String)] -> m String
42 invokePostService url args = do
43     let args1 = map (\(x, y) -> BC.pack x, BC.pack y)) args
44     r <- liftSIO $ C.postForm (BC.pack url) args1 getResult
45     eitherSIO r
46     where getResult :: C.Response -> IOI.InputStream BC.ByteString ->
47             IO (Either ServiceError String)
48     getResult _ s = do
49         bs <- IOI.read s
50         case bs of
51             Nothing ->
52                 return $ Left (InternalFailure "unexpected service response")
53             Just bs1 -> let b = BC.unpack bs1 in
54                 if take 6 b == "error:"
55                 then return $ Left (InternalFailure b)
56                 else return $ Right b

```

LISTING D.18: Services.hs

```

1 module SeqActIO (SeqActIO, runSeqActIO) where
2 import Internal.ActIO

```

LISTING D.19: SeqActIO.hs

```

1 {-# LANGUAGE OverloadedStrings, ScopedTypeVariables #-}
2
3 module Main (main) where
4
5 import Control.Monad.IO.Class (MonadIO, liftIO)
6 import qualified Data.Text as T
7 import Web.Spock.Safe
8 import Control.Monad.Trans.Either

```

```
9 import Text.Read (readMaybe)
10 import Control.Monad.Trans.Class (lift)
11
12 maybeError :: String -> Maybe a -> Either String a
13 maybeError e x = case x of
14     Nothing -> Left e
15     Just x1 -> Right x1
16
17 getArg :: (MonadIO m) => String -> ActionCtxT ctx m (Either String String)
18 getArg p = do
19     t <- param (T.pack p)
20     return $ maybeError ("missing '"++p++"' argument") t
21
22 main :: IO ()
23 main = runSpock 5002 . spockT id $
24     post "/add" addService
25
26 getLeftRight :: (MonadIO m) => ActionCtxT ctx m (Either String (Double,Double))
27 getLeftRight = runEitherT $ do
28     lef <- lift (getArg "left") >>= hoistEither
29     rig <- lift (getArg "right") >>= hoistEither
30     let x = readMaybe lef
31     let y = readMaybe rig
32     x1 <- case x of
33         Nothing -> hoistEither (Left "invalid 'left' argument")
34         Just i -> return i
35     y1 <- case y of
36         Nothing -> hoistEither (Left "invalid 'right' argument")
37         Just i -> return i
38     return (x1,y1)
39
40 addService :: (MonadIO m) => ActionCtxT ctx m ()
41 addService = do
42     liftIO $ putStrLn ("arithservice: invoke add")
43     ii <- getLeftRight
44     case ii of
45         Left e -> do
46             liftIO $ putStrLn ("arithservice: add error " ++ e)
47             text (T.pack $ "error: " ++ e)
48         Right (x,y) -> do
49             liftIO $ putStrLn ("arithservice: add result " ++ show (x+y))
50             text (T.pack (show $ x+y))
51
52 multiplyService :: (MonadIO m) => ActionCtxT ctx m ()
53 multiplyService = do
54     liftIO $ putStrLn ("arithservice: invoke multiply")
55     ii <- getLeftRight
56     case ii of
57         Left e -> do
58             liftIO $ putStrLn ("arithservice: multiply error " ++ e)
59             text (T.pack $ "error: " ++ e)
60         Right (x,y) -> do
61             liftIO $ putStrLn ("arithservice: multiply result " ++ show (x*y))
62             text (T.pack (show $ x*y))
63
```

```
64 minusService :: (MonadIO m) => ActionCtxT ctx m ()
65 minusService = do
66     liftIO $ putStrLn ("arithservice: invoke minus")
67     ii <- getLeftRight
68     case ii of
69         Left e -> do
70             liftIO $ putStrLn ("arithservice: minus error " ++ e)
71             text (T.pack $ "error: " ++ e)
72         Right (x,y) -> do
73             liftIO $ putStrLn ("arithservice: minus result " ++ show (x-y))
74             text (T.pack (show $ x-y))
```

LISTING D.20: ArithServices.hs

Appendix E

Java Bank Integrated Development Environment

```
1 package util;
2
3 import java.util.regex.Pattern;
4
5 public class UtilRegex {
6     public static boolean isIdentifier(String id) {
7         return !Pattern.matches("^[0-9].*$", id) && Pattern.matches("^\w+$", id);
8     }
9 }
```

LISTING E.1: util/UtilRegex.java

```
1 package util;
2
3 public class StringUtil {
4     public static int countSubstring(String str, String substr) {
5         return (str.length() - str.replace(substr, "").length()) / substr.length();
6     }
7 }
```

LISTING E.2: util/StringUtil.java

```
1 package util;
2
3 import java.awt.Component;
4
5 import javax.swing.JOptionPane;
6
7 public class InputDialog {
8     public static String show(Component parent, String message) {
9         return JOptionPane.showInputDialog(parent, message);
10    }
11 }
```

LISTING E.3: util/InputDialog.java

```
1 package util;
2
3 import java.awt.Insets;
4
5 import javax.swing.JButton;
6
7 @SuppressWarnings("serial")
8 public class UtilButton extends JButton {
9     private final static Insets margin = new Insets(0, 5, 0, 5);
10    public UtilButton(String text) {
11        super(text);
12        setMargin(margin);
13       setFont(UtilFont.textFont);
14    }
15 }
```

LISTING E.4: util/UtilButton.java

```
1 package util;
2
3 import java.awt.Component;
4
5 import javax.swing.JScrollPane;
6
7 @SuppressWarnings("serial")
8 public class UtilScrollPane extends JScrollPane {
9     private final static int SCROLL_INCREMENT = 15;
10
11    public UtilScrollPane(Component c) {
12        super(c);
13        getVerticalScrollBar().setUnitIncrement(SCROLL_INCREMENT);
14    }
15
16    public UtilScrollPane(Component c, int hsbPolity, int vsbPolicy) {
17        super(c, hsbPolity, vsbPolicy);
18        getVerticalScrollBar().setUnitIncrement(SCROLL_INCREMENT);
19    }
20
21    public UtilScrollPane() {
22        getVerticalScrollBar().setUnitIncrement(SCROLL_INCREMENT);
23    }
24 }
```

LISTING E.5: util/UtilScrollPane.java

```
1 package util;
2
3 import java.awt.Color;
4
5 public class ColorTheme {
```

```

6     public final static Color HOVERED = new Color(0x77, 0x77, 0x77, 127);
7     public final static Color SELECTED = new Color(0xCC, 0x66, 0, 127);
8
9     public final static Color TRANS_YELLOW = new Color(0xFF, 0xFF, 0x33, 127);
10    public final static Color TRANS_GREEN = new Color(0x32, 0xCD, 0x32, 127);
11    public final static Color TRANS_BLUE = new Color(0x1f, 0xbe, 0xd6, 127);
12    public final static Color TRANS_RED = new Color(0xff, 0x99, 0x99, 127);
13    public final static Color TRANS_LIGHT_GRAY = new Color(0xCC, 0xCC, 0xCC, 127);
14
15    public static final Color NONTRANS_RED = new Color(0xff, 0x99, 0x99);
16    public final static Color NONTRANS_YELLOW = new Color(0xFF, 0xFF, 0x33);
17    public final static Color NONTRANS_GREEN = new Color(0x32, 0xCD, 0x32);
18    public final static Color NONTRANS_BLUE = new Color(0x1f, 0xbe, 0xd6);
19    public final static Color NONTRANS_LIGHT_GRAY = new Color(0xCC, 0xCC, 0xCC);
20
21    public final static Color DARK_YELLOW = new Color(0xFF, 0xFF, 0x33);
22    public final static Color DARK_GREEN = new Color(0x22, 0x8b, 0x22);
23    public final static Color DARK_BLUE = new Color(0x22, 0x22, 0x8b);
24    public final static Color DARK_RED = new Color(0x8b, 0x22, 0x22);
25
26 }

```

LISTING E.6: util/ColorTheme.java

```

1 package util;
2
3 import java.awt.Color;
4 import java.awt.Font;
5
6 import javax.swing.text.JTextComponent;
7
8 import org.jdesktop.swingx.PromptSupport;
9
10 public class UtilPrompt {
11     public static void setPrompt(String prompt, JTextComponent comp) {
12         PromptSupport.setPrompt(prompt, comp);
13         PromptSupport.setFontStyle(Font.ITALIC, comp);
14         PromptSupport.setForeground(Color.gray, comp);
15     }
16 }

```

LISTING E.7: util/UtilPrompt.java

```

1 package util;
2
3 import java.awt.Font;
4
5 public class UtilFont {
6     public static final String fontName = "SansSerif";
7     public static final Font titleFont = new Font(fontName, Font.BOLD, 12);
8     public static final Font textFont = new Font(fontName, Font.PLAIN, 12);
9     public static final Font smallItalicFont = new Font(fontName, Font.ITALIC, 11);
10    public static final Font boldTextFont = new Font(fontName, Font.BOLD, 12);
11    public static final Font errorTextFont = new Font(fontName, Font.ITALIC, 12);

```

12 }

LISTING E.8: util/UtilFont.java

```
1 package util;
2
3 import javax.swing.JLabel;
4
5 @SuppressWarnings("serial")
6 public class UtilLabel extends JLabel {
7     public UtilLabel(String text) {
8         super(text);
9         setFont(UtilFont.textFont);
10    }
11 }
```

LISTING E.9: util/UtilLabel.java

```
1 package util;
2
3 import javax.swing.ImageIcon;
4
5 public class IconUtil {
6     public static ImageIcon createImageIcon(String path) {
7         IconUtil ut = new IconUtil();
8         java.net.URL imgURL = ut.getClass().getResource("../" + path);
9         if (imgURL != null) {
10             return new ImageIcon(imgURL, "icon");
11         } else {
12             System.err.println("Couldn't find file: " + path);
13             return null;
14         }
15     }
16 }
```

LISTING E.10: util/IconUtil.java

```
1 package lang.constant;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import lang.type.RecordColumn;
7 import lang.type.Type;
8 import lang.type.TypeRecord;
9
10 public class ConstantRecord implements Constant {
11     private TypeRecord type;
12     private List<RecordValue> columnsInOrder;
13
14     public ConstantRecord(List<RecordValue> columns) {
15         columnsInOrder = new ArrayList<>(columns);
16         ArrayList<RecordColumn> typeCols = new ArrayList<>();
```

```
17     for (RecordValue v : columns)
18         typeCols.add(new RecordColumn(v.getId(), v.getConstant().getType()));
19     type = new TypeRecord(typeCols);
20 }
21
22 public List<RecordValue> getColumnsInOrder() {
23     return new ArrayList<>(columnsInOrder);
24 }
25
26 @Override
27 public Type getType() {
28     return type;
29 }
30
31 @Override
32 public boolean equals(Object x) {
33     if (!(x instanceof ConstantRecord))
34         return false;
35     ConstantRecord other = (ConstantRecord)x;
36     if (other.columnsInOrder.size() != columnsInOrder.size())
37         return false;
38
39     for (int i = 0; i < columnsInOrder.size(); i++) {
40         if (!columnsInOrder.get(i).equals(other.columnsInOrder.get(i)))
41             return false;
42     }
43     return true;
44 }
45
46 @Override
47 public int hashCode() {
48     return columnsInOrder.stream()
49         .map(Object::hashCode)
50         .reduce((x,y) -> x+y).get();
51 }
52
53 @Override
54 public String toString() {
55     StringBuilder b = new StringBuilder("Record(");
56     for (int i = 0; i < columnsInOrder.size()-1; i++) {
57         b.append(columnsInOrder.get(i));
58         b.append(", ");
59     }
60     b.append(columnsInOrder.get(columnsInOrder.size()-1));
61     b.append(')');
62     return b.toString();
63 }
64
65 @Override
66 public String getShortDescription() {
67     return "Record(...)";
68 }
69 }
```

LISTING E.11: lang/constant/ConstantRecord.java

```

1 package lang.constant;
2
3 public class RecordValue {
4     private String id;
5     private Constant constant;
6
7     public RecordValue(String id, Constant constant) {
8         this.id = id;
9         this.constant = constant;
10    }
11
12    public String getId() {
13        return id;
14    }
15
16    public Constant getConstant() {
17        return constant;
18    }
19
20    @Override
21    public int hashCode() {
22        return id.hashCode() ^ constant.hashCode();
23    }
24
25    @Override
26    public boolean equals(Object x) {
27        if (!(x instanceof RecordValue))
28            return false;
29        return ((RecordValue)x).id.equals(id) && ((RecordValue)x).constant.equals(constant);
30    }
31
32    @Override
33    public String toString() {
34        return id + ": " + constant;
35    }
36 }
```

LISTING E.12: lang/constant/RecordValue.java

```

1 package lang.constant;
2
3 @SuppressWarnings("serial")
4 public class InvalidConstantException extends Exception {
5     public InvalidConstantException(String message) {
6         super(message);
7     }
8 }
```

LISTING E.13: lang/constant/InvalidConstantException.java

```

1 package lang.constant;
2
3 import lang.type.Type;
```

```
4 import lang.type.TypeTime;
5
6 public class ConstantTime implements Constant {
7     private TypeTime type = new TypeTime();
8     private Long value;
9
10    public ConstantTime(long value) {
11        this.value = value;
12    }
13
14    @Override
15    public Type getType() {
16        return type;
17    }
18
19    @Override
20    public boolean equals(Object x) {
21        if (!(x instanceof ConstantTime))
22            return false;
23        ConstantTime other = (ConstantTime)x;
24        return other.value.equals(value);
25    }
26
27    @Override
28    public int hashCode() {
29        return value.hashCode() ^ type.hashCode();
30    }
31
32    @Override
33    public String toString() {
34        return getShortDescription();
35    }
36
37    public static ConstantTime parse(String str) throws InvalidConstantException {
38        throw new InvalidConstantException("Invalid time '"+str+"'");
39    }
40
41    @Override
42    public String getShortDescription() {
43        return value.toString();
44    }
45 }
```

LISTING E.14: lang/constant/ConstantTime.java

```
1 package lang.constant;
2
3 import lang.type.Type;
4 import lang.type.TypeCode;
5
6 public interface Constant {
7     public Type getType();
8
9     public String getShortDescription();
10
```

```
11     default public TypeCode getTypeCode() {
12         return getType().getTypeCode();
13     }
14 }
```

LISTING E.15: lang/constant/Constant.java

```
1 package lang.constant;
2
3 import lang.type.Type;
4 import lang.type.TypeBool;
5
6 public class ConstantBool implements Constant {
7     private TypeBool type = new TypeBool();
8     private Boolean value;
9
10    public ConstantBool(boolean value) {
11        this.value = value;
12    }
13
14    public Boolean getValue() {
15        return value;
16    }
17
18    @Override
19    public Type getType() {
20        return type;
21    }
22
23    @Override
24    public boolean equals(Object x) {
25        if (!(x instanceof ConstantBool))
26            return false;
27        ConstantBool other = (ConstantBool)x;
28        return other.value.equals(value);
29    }
30
31    @Override
32    public int hashCode() {
33        return value.hashCode() ^ type.hashCode();
34    }
35
36    @Override
37    public String getShortDescription() {
38        return value ? "True" : "False";
39    }
40
41    @Override
42    public String toString() {
43        return getShortDescription();
44    }
45
46    public static ConstantBool parse(String str) throws InvalidConstantException {
47        if (str.equals("True"))
48            return new ConstantBool(true);
```

```
49     else if (str.equals("False"))
50         return new ConstantBool(false);
51     throw new InvalidConstantException("Invalid boolean '"+str+"'");
52 }
53 }
```

LISTING E.16: lang/constant/ConstantBool.java

```
1 package lang.constant;
2
3 import java.math.BigDecimal;
4
5 import lang.type.Type;
6 import lang.type.TypeNumber;
7
8 public class ConstantNumber implements Constant {
9     private TypeNumber type = new TypeNumber();
10    private BigDecimal value;
11
12    public ConstantNumber(BigDecimal value) {
13        this.value = value;
14    }
15
16    public BigDecimal getValue() {
17        return value;
18    }
19
20    @Override
21    public Type getType() {
22        return type;
23    }
24
25    @Override
26    public boolean equals(Object x) {
27        if (!(x instanceof ConstantNumber))
28            return false;
29        ConstantNumber other = (ConstantNumber)x;
30        return other.value.equals(value);
31    }
32
33    @Override
34    public int hashCode() {
35        return value.hashCode() ^ type.hashCode();
36    }
37
38    @Override
39    public String toString() {
40        return getShortDescription();
41    }
42
43    public static ConstantNumber parse(String str) throws InvalidConstantException {
44        try {
45            return new ConstantNumber(new BigDecimal(str));
46        } catch (NumberFormatException e) {
47            throw new InvalidConstantException("Invalid number '"+str+"');");
48    }
49}
```

```
48     }
49 }
50
51 @Override
52 public String getShortDescription() {
53     return value.toString();
54 }
55 }
```

LISTING E.17: lang/constant/ConstantNumber.java

```
1 package lang.constant;
2
3 import lang.type.Type;
4 import lang.type.TypeString;
5
6 public class ConstantString implements Constant {
7     private TypeString type = new TypeString();
8     private String value;
9
10    public ConstantString(String value) {
11        this.value = value;
12    }
13
14    public String getValue() {
15        return value;
16    }
17
18    @Override
19    public Type getType() {
20        return type;
21    }
22
23    @Override
24    public boolean equals(Object x) {
25        if (!(x instanceof ConstantString))
26            return false;
27        ConstantString other = (ConstantString)x;
28        return other.value.equals(value);
29    }
30
31    @Override
32    public int hashCode() {
33        return value.hashCode() ^ type.hashCode();
34    }
35
36    @Override
37    public String toString() {
38        return value;
39    }
40
41    public static ConstantString parse(String str) {
42        return new ConstantString(str);
43    }
44}
```

```
45     @Override
46     public String getShortDescription() {
47         String str = value;
48         if (str.contains("\n"))
49             str = str.split("\n", 2)[0] + "...";
50         if (str.length() > 22) {
51             str = str.substring(0, 22) + "...";
52         }
53         return str;
54     }
55 }
```

LISTING E.18: lang/constant/ConstantString.java

```
1 package lang.constant;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import lang.type.Type;
7 import lang.type.TypeList;
8
9 public class ConstantList implements Constant {
10     private Type type;
11     private List<Constant> values;
12
13     public ConstantList(List<Constant> values, Type childType) {
14         type = new TypeList(childType);
15         this.values = new ArrayList<>(values);
16         for (Constant c : values) {
17             if (!c.getType().equals(childType))
18                 throw new IllegalArgumentException("Inconsistent list types");
19         }
20     }
21
22     public List<Constant> getValues() {
23         return values;
24     }
25
26     @Override
27     public Type getType() {
28         return type;
29     }
30
31     @Override
32     public boolean equals(Object x) {
33         if (!(x instanceof ConstantList))
34             return false;
35         ConstantList other = (ConstantList)x;
36         if (values.size() != other.values.size())
37             return false;
38         for (int i = 0; i < values.size(); i++) {
39             if (!values.get(i).equals(other.values.get(i)))
40                 return false;
41         }
42     }
43 }
```

```
42         return true;
43     }
44
45     @Override
46     public int hashCode() {
47         return values.stream()
48             .map((c) -> c.hashCode())
49             .reduce((x,y) -> x+y)
50             .orElse(type.hashCode());
51     }
52
53     @Override
54     public String toString() {
55         StringBuilder b = new StringBuilder();
56         b.append("List(");
57         for (int i = 0; i < values.size()-1; i++) {
58             b.append(values.get(i));
59             b.append(", ");
60         }
61         if (values.size() > 0)
62             b.append(values.get(values.size()-1));
63         b.append(')');
64         return b.toString();
65     }
66
67     @Override
68     public String getShortDescription() {
69         return "List(...)";
70     }
71 }
```

LISTING E.19: lang/constant/ConstantList.java

```
1 package lang.ast;
2
3 import java.awt.Point;
4 import java.util.List;
5 import java.util.Map;
6
7 import lang.type.RecordColumn;
8
9 public class ASTCFComponentIf extends ASTCFComponent {
10     String ifIdentifier;
11
12     public ASTCFComponentIf(Point location, List<RecordColumn> inputs, Map<Integer,String>
13         eventMap, int seqNum) {
14         super(location, inputs, eventMap, seqNum);
15         ifIdentifier = ASTUtil.nextUniqueIdentifier();
16     }
17
18     @Override
19     public void accept(ASTVisitor v) {
20         v.visit(this);
21     }
22 }
```

```
22     @Override
23     String getId() {
24         return identifier;
25     }
26
27     @Override
28     public String getTitle() {
29         return "If";
30     }
31 }
```

LISTING E.20: lang/ast/ASTCFComponentIf.java

```
1 package lang.ast;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import lang.type.RecordColumn;
7
8 public class ASTDF implements ASTNode {
9     String dataFlowTitle;
10    List<RecordColumn> dataFlowInputs;
11    List<RecordColumn> dataFlowOutputs;
12    List<ASTDFComponent> dataFlowComponents;
13    List<ASTDFEdge> dataFlowEdges;
14    boolean dataFlowInputOrderFixed;
15
16    public ASTDF(String title, List<RecordColumn> inputs, List<RecordColumn> outputs, boolean
17        keepInputOrder) {
18        dataFlowTitle = title;
19        dataFlowInputs = new ArrayList<>(inputs);
20        dataFlowOutputs = new ArrayList<>(outputs);
21        dataFlowComponents = new ArrayList<>();
22        dataFlowEdges = new ArrayList<>();
23        dataFlowInputOrderFixed = keepInputOrder;
24    }
25
26    @Override
27    public void accept(ASTVisitor v) {
28        v.visit(this);
29    }
30
31    public void addComponent(ASTDFComponent c) {
32        dataFlowComponents.add(c);
33    }
34
35    public void addAllComponents(List<ASTDFComponent> cs) {
36        dataFlowComponents.addAll(cs);
37    }
38
39    public void addDataFlowEdge(ASTDFEdge edge) {
40        dataFlowEdges.add(edge);
41    }
42}
```

```

42     public void addAllDataFlowEdges(List<ASTDFEdge> edges) {
43         dataFlowEdges.addAll(edges);
44     }
45 }
```

LISTING E.21: lang/ast/ASTDF.java

```

1 package lang.ast;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class ASTCF implements ASTNode {
7     String controlFlowTitle;
8     String controlFlowPlatform;
9     String controlFlowGroup;
10    List<ASTCFComponent> components = new ArrayList<>();
11    List<ASTCFEdge> connections = new ArrayList<>();
12
13    public ASTCF(String title, String platform, String group) {
14        controlFlowTitle = title;
15        controlFlowPlatform = platform;
16        controlFlowGroup = group;
17    }
18
19    public void addComponent(ASTCFComponent comp) {
20        components.add(comp);
21    }
22
23    public void addEdge(ASTCFEdge conn) {
24        connections.add(conn);
25    }
26
27    @Override
28    public void accept(ASTVisitor v) {
29        v.visit(this);
30    }
31 }
```

LISTING E.22: lang/ast/ASTCF.java

```

1 package lang.ast;
2
3 import java.awt.Point;
4 import java.util.List;
5
6 public class ASTDFComponentArithmetic extends ASTDFComponent {
7     String arithmeticId;
8     String arithmeticType;
9
10    public ASTDFComponentArithmetic(Point location, String type, String title,
11                                    List<String> inputs, List<String> outputs, int seqNumber) {
12        super(location, title, inputs, outputs, seqNumber);
13        arithmeticId = ASTUtil.nextUniqueIdentifier();
```

```
14     this.arithmeticType = type;
15 }
16
17 @Override
18 public void accept(ASTVisitor v) {
19     v.visit(this);
20 }
21
22 @Override
23 String getId() {
24     return arithmeticId;
25 }
26 }
```

LISTING E.23: lang/ast/ASTDFComponentArithmetic.java

```
1 package lang.ast;
2
3 import java.awt.Point;
4 import java.util.ArrayList;
5 import java.util.List;
6 import java.util.Map;
7
8 import lang.type.RecordColumn;
9
10 public class ASTCFComponentInteraction extends ASTCFComponent {
11     String interactionId;
12     List<RecordColumn> interactionOutputs;
13     String interactionTitle;
14     String interactionPlatform;
15
16     public ASTCFComponentInteraction(Point location, String title, String platform, List<
17         RecordColumn> inputs,
18         List<RecordColumn> normalOutputs, Map<Integer, String> eventMap,
19         int seqNum) {
20         super(location, inputs, eventMap, seqNum);
21         interactionId = ASTUtil.nextUniqueIdentifier();
22         interactionOutputs = new ArrayList<>(normalOutputs);
23         interactionTitle = title;
24         interactionPlatform = platform;
25     }
26
27     @Override
28     public void accept(ASTVisitor v) {
29         v.visit(this);
30     }
31
32     @Override
33     String getId() {
34         return interactionId;
35     }
36
37     @Override
38     String getTitle() {
39         return interactionTitle;
```

```
39     }
40
41 }
```

LISTING E.24: lang/ast/ASTCFComponentInteraction.java

```
1 package lang.ast;
2
3 import java.awt.Point;
4 import java.util.List;
5
6 public class ASTDFComponentStringFormat extends ASTDFComponent {
7     String stringFormatId;
8     String stringFormat;
9
10    public ASTDFComponentStringFormat(String format, Point location, String title, List<String>
11        inputs, List<String> outputs, int seqNumber) {
12        super(location, title, inputs, outputs, seqNumber);
13        stringFormat = format;
14        stringFormatId = ASTUtil.nextUniqueIdentifier();
15    }
16
17    @Override
18    public void accept(ASTVisitor v) {
19        v.visit(this);
20    }
21
22    @Override
23    String getId() {
24        return stringFormatId;
25    }
26}
```

LISTING E.25: lang/ast/ASTDFComponentStringFormat.java

```
1 package lang.ast;
2
3 import java.awt.Point;
4 import java.util.List;
5
6 import lang.type.Type;
7
8 public class ASTDFComponentSink extends ASTDFComponent {
9     String sinkId;
10    Type sinkType;
11
12    public ASTDFComponentSink(Point location, Type type, List<String> inputs, List<String> outputs
13        , int seqNumber) {
14        super(location, "(Sink)", inputs, outputs, seqNumber);
15        sinkId = ASTUtil.nextUniqueIdentifier();
16        sinkType = type;
17    }
18}
```

```

18     @Override
19     public void accept(ASTVisitor v) {
20         v.visit(this);
21     }
22
23     @Override
24     String getId() {
25         return sinkId;
26     }
27
28 }
```

LISTING E.26: lang/ast/ASTDFComponentSink.java

```

1 package lang.ast;
2
3 import java.awt.Point;
4 import java.util.List;
5
6 import lang.type.Type;
7
8 public class ASTDFComponentOutput extends ASTDFComponent {
9     String outputId;
10    Type outputOriginalType;
11    Type outputType;
12    String outputTitle;
13
14    public ASTDFComponentOutput(Point location, Type orig, Type type, String name,
15        List<String> inputs, List<String> outputs, int seqNumber) {
16        super(location, name, inputs, outputs, seqNumber);
17        outputId = ASTUtil.nextUniqueIdentifier();
18        outputOriginalType = orig;
19        outputType = type;
20        outputTitle = name;
21    }
22
23    @Override
24    public void accept(ASTVisitor v) {
25        v.visit(this);
26    }
27
28    @Override
29    String getId() {
30        return outputId;
31    }
32
33 }
```

LISTING E.27: lang/ast/ASTDFComponentOutput.java

```

1 package lang.ast;
2
3 import java.awt.Point;
4 import java.util.ArrayList;
```

```

5 import java.util.List;
6
7 import lang.type.Type;
8
9 public class ASTDFComponentListOperation extends ASTDFComponent {
10     String listOperationId;
11     String listOperationType;
12     List<Type> listOperationOriginalTypes;
13     List<Type> listOperationTypes;
14
15     public ASTDFComponentListOperation(Point location, String type, List<Type> origTypes,
16             List<Type> opTypes, String title, List<String> inputs, List<String> outputs, int
17             seqNumber) {
18         super(location, title, inputs, outputs, seqNumber);
19         listOperationId = ASTUtil.nextUniqueIdentifier();
20         listOperationType = type;
21         listOperationTypes = new ArrayList<>(opTypes);
22         listOperationOriginalTypes = new ArrayList<>(origTypes);
23     }
24
25     @Override
26     public void accept(ASTVisitor v) {
27         v.visit(this);
28     }
29
30     @Override
31     String getId() {
32         return listOperationId;
33     }
34 }
```

LISTING E.28: lang/ast/ASTDFComponentListOperation.java

```

1 package lang.ast;
2
3 import java.io.PrintStream;
4 import java.util.HashSet;
5
6 import org.apache.commons.lang3.StringEscapeUtils;
7
8 import gui.controlflow.ControlPart;
9 import lang.constant.Constant;
10 import lang.constant.ConstantBool;
11 import lang.constant.ConstantList;
12 import lang.constant.ConstantNumber;
13 import lang.constant.ConstantRecord;
14 import lang.constant.ConstantString;
15 import lang.constant.RecordValue;
16 import lang.type.RecordColumn;
17 import lang.type.Type;
18 import lang.type.TypeAuto;
19 import lang.type.TypeBool;
20 import lang.type.TypeConnection;
21 import lang.type.TypeError;
```

```
22 import lang.type.TypeList;
23 import lang.type.TypeNumber;
24 import lang.type.TypeRecord;
25 import lang.type.TypeString;
26 import lang.type.TypeTime;
27 import lang.type.TypeUnknown;
28
29 public class ASTVisitorPrint implements ASTVisitor {
30     private final static int indentationSize = 2;
31
32     private int indentation = 0;
33     private PrintStream outputStream;
34
35     private void incIndentation() {
36         indentation += indentationSize;
37     }
38
39     private void decIndentation() {
40         indentation -= indentationSize;
41     }
42
43     public ASTVisitorPrint(PrintStream stream) {
44         outputStream = stream;
45     }
46
47     private void printfln(String fmt, Object... args) {
48         for (int i = 0; i < indentation; i++)
49             outputStream.print(' ');
50         outputStream.printf(fmt, args);
51         outputStream.println();
52     }
53
54     @Override
55     public void visit(ASTCF n) {
56         printfln("<ControlFlow id=\"%s\" platform=\"%s\" group=\"%s\">",
57                 n.controlFlowTitle, n.controlFlowPlatform, n.controlFlowGroup);
58         incIndentation();
59
60         printfln("<Components>");
61         incIndentation();
62         for (ASTCFComponent c : n.components) {
63             c.accept(this);
64         }
65         decIndentation();
66         printfln("</Components>");
67
68         printfln("<Edges>");
69         incIndentation();
70         for (ASTCFEdge c : n.connections) {
71             c.accept(this);
72         }
73         decIndentation();
74         printfln("</Edges>");
75
76         decIndentation();
```

```
77         printfln("</ControlFlow>");
78     }
79
80     @Override
81     public void visit(ASTCFTypeConnection n) {
82         printfln("<TypeConnection outputName=\"%s\" outputComponent=\"%s\" inputName=\"%s\""
83             inputComponent=\"%s\">",
84             n.outputName, n.outputComponent.getId(), n.inputName, n.inputComponent.getId());
85         incIndentation();
86
87         printfln("<Connections>");
88         incIndentation();
89         for (TypeConnection.SingleConnection c : n.typeConnection.getConnections()) {
90             printfln("<Connection outputIndex=\"%d\" inputIndex = \"%d\"></Connection>",
91                 c.getOutputIndex(), c.getInputIndex());
92         }
93         decIndentation();
94         printfln("</Connections>");
95
96         decIndentation();
97         printfln("</TypeConnection>");
98     }
99
100    @Override
101    public void visit(ASTCFComponentInteraction n) {
102        printfln("<InteractionComponent id=\"%s\" title=\"%s\" platform=\"%s\" x=\"%d\" y=\"%d\""
103             seq=\"%d\">",
104             n.interactionId, n.interactionTitle, n.interactionPlatform, n.componentLocation.x,
105             n.componentLocation.y, n.sequenceNumber);
106        incIndentation();
107
108        printfln("<InteractionEvents>");
109        incIndentation();
110        for (String e : new HashSet<>(n.componentEventMap.values())) {
111            if (e.equals(ControlPart.errorEvent))
112                continue;
113            printfln("<InteractionEvent id=\"%s\"></InteractionEvent>", e);
114        }
115        decIndentation();
116        printfln("</InteractionEvents>");
117
118        printfln("<InteractionInputs>");
119        incIndentation();
120        for (RecordColumn col : n.componentInputs) {
121            printfln("<InteractionInput id=\"%s\">", col.getId());
122            incIndentation();
123            printType(col.getType());
124            decIndentation();
125            printfln("</InteractionInput>");
126        }
127        decIndentation();
128        printfln("</InteractionInputs>");
```

```
129     for (RecordColumn col : n.interactionOutputs) {
130         printfln("<InteractionOutput id=\"%s\">", col.getId());
131         incIndentation();
132         printType(col.getType());
133         decIndentation();
134         printfln("</InteractionOutput>");
135     }
136     decIndentation();
137     printfln("</InteractionOutputs>");
138
139     decIndentation();
140     printfln("</InteractionComponent>");
141 }
142
143 @Override
144 public void visit(ASTCFComponentDF n) {
145     printfln("<DataFlowComponent id=\"%s\" x=\"%d\" y=\"%d\" seq=\"%d\" canFail=\"%s\">",
146             n.dataFlow.dataFlowTitle,
147             n.componentLocation.x,
148             n.componentLocation.y,
149             n.sequenceNumber,
150             n.dataFlowCanFail ? "1" : "0");
151     incIndentation();
152     n.dataFlow.accept(this);
153     decIndentation();
154     printfln("</DataFlowComponent>");
155 }
156
157 @Override
158 public void visit(ASTCFComponentIf n) {
159     printfln("<If id=\"%s\" x=\"%d\" y=\"%d\" seq=\"%d\"></If>",
160             n.getId(),
161             n.componentLocation.x,
162             n.componentLocation.y,
163             n.sequenceNumber);
164 }
165
166 @Override
167 public void visit(ASTCFComponentFail n) {
168     printfln("<Fail id=\"%s\" x=\"%d\" y=\"%d\" seq=\"%d\"></Fail>",
169             n.getId(),
170             n.componentLocation.x,
171             n.componentLocation.y,
172             n.sequenceNumber);
173 }
174
175 @Override
176 public void visit(ASTCFComponentStop n) {
177     printfln("<Stop id=\"%s\" x=\"%d\" y=\"%d\" seq=\"%d\"></Stop>",
178             n.getId(),
179             n.componentLocation.x,
180             n.componentLocation.y,
181             n.sequenceNumber);
182 }
183
```

```
184     @Override
185     public void visit(ASTCFComponentStart n) {
186         printfln("<Start id=\"%s\" x=\"%d\" y=\"%d\" seq=\"%d\"></Start>",
187                 n.getId(),
188                 n.componentLocation.x,
189                 n.componentLocation.y,
190                 n.sequenceNumber);
191     }
192
193     @Override
194     public void visit(ASTCFEdge n) {
195         printfln("<Edge eventIndex=\"%d\" start=\"%s\" end = \"%s\">",
196                 n.startEventIndex,
197                 n.startComponent.getId(),
198                 n.endComponent.getId());
199         incIndentation();
200         for (ASTCFTypeConnection c : n.typeConnections) {
201             c.accept(this);
202         }
203         decIndentation();
204         printfln("</Edge>");
205     }
206
207     private void printConstant(Constant t) {
208         switch (t.getTypeCode()) {
209             case BOOL:
210                 printBoolConstant((ConstantBool)t);
211                 break;
212             case LIST:
213                 printListConstant((ConstantList)t);
214                 break;
215             case NUMBER:
216                 printNumberConstant((ConstantNumber)t);
217                 break;
218             case RECORD:
219                 printRecordConstant((ConstantRecord)t);
220                 break;
221             case STRING:
222                 printStringConstant((ConstantString)t);
223                 break;
224             case TIME:
225             case ERROR:
226             case UNKNOWN:
227                 default:
228                     throw new RuntimeException();
229                 }
230     }
231
232     private void printStringConstant(ConstantString c) {
233         String a = StringEscapeUtils.escapeXml11(c.toString());
234         String b = a.replaceAll("\n", "\r\n");
235         printfln("<ConstantString value=\"%s\"></ConstantString>", b);
236     }
237
238     private void printRecordConstant(ConstantRecord c) {
```

```
239     printfln("<ConstantRecord>");
240     incIndentation();
241     for (RecordValue v : c.getColumnsInOrder()) {
242         printfln("<RecordValue id=\"%s\">", v.getId());
243         incIndentation();
244         printConstant(v.getConstant());
245         decIndentation();
246         printfln("</RecordValue>");
247     }
248     decIndentation();
249     printfln("</ConstantRecord>");
250 }
251
252 private void printNumberConstant(ConstantNumber c) {
253     printfln("<ConstantNumber value=\"%s\"></ConstantNumber>", c.toString());
254 }
255
256 private void printListConstant(ConstantList c) {
257     printfln("<ConstantList>");
258     incIndentation();
259     printType(c.getType());
260     for (Constant val : c.getValues()) {
261         printfln("<ListItem>");
262         incIndentation();
263         printConstant(val);
264         decIndentation();
265         printfln("</ListItem>");
266     }
267     decIndentation();
268     printfln("</ConstantList>");
269 }
270
271 private void printBoolConstant(ConstantBool c) {
272     printfln("<ConstantBool value=\"%s\"></ConstantBool>", c.toString());
273 }
274
275 private void printType(Type t) {
276     switch (t.getTypeCode()) {
277     case BOOL:
278         printTypeBool((TypeBool)t);
279         break;
280     case ERROR:
281         printTypeError((TypeError)t);
282         break;
283     case LIST:
284         printTypeList((TypeList)t);
285         break;
286     case NUMBER:
287         printTypeNumber((TypeNumber)t);
288         break;
289     case RECORD:
290         printTypeRecord((TypeRecord)t);
291         break;
292     case STRING:
293         printTypeString((TypeString)t);
```

```
294         break;
295     case TIME:
296         printTypeTime((TypeTime)t);
297         break;
298     case AUTO:
299         printTypeAuto((TypeAuto)t);
300         break;
301     case UNKNOWN:
302         printTypeUnknown((TypeUnknown)t);
303         break;
304     default:
305         throw new RuntimeException();
306     }
307 }
308
309 private void printTypeUnknown(TypeUnknown t) {
310     printfln("<TypeUnknown message=\"%s\"></TypeUnknown>", t.getMessage());
311 }
312
313 private void printTypeTime(TypeTime t) {
314     printfln("<TypeTime></TypeTime>");
315 }
316
317 private void printTypeString(TypeString t) {
318     printfln("<TypeString></TypeString>");
319 }
320
321 private void printTypeRecord(TypeRecord t) {
322     printfln("<TypeRecord>");
323     incIndentation();
324     for (RecordColumn col : t.getColumnsInOrder()) {
325         printfln("<RecordColumn id=\"%s\">", col.getId());
326         incIndentation();
327         printType(col.getType());
328         decIndentation();
329         printfln("</RecordColumn>");
330     }
331     decIndentation();
332     printfln("</TypeRecord>");
333 }
334
335 private void printTypeNumber(TypeNumber t) {
336     printfln("<TypeNumber></TypeNumber>");
337 }
338
339 private void printTypeAuto(TypeAuto t) {
340     printfln("<TypeAuto>");
341     incIndentation();
342     printType(t.getType());
343     decIndentation();
344     printfln("</TypeAuto>");
345 }
346
347 private void printTypeList(TypeList t) {
348     printfln("<TypeList>");
```

```
349     incIndentation();
350     printType(t.getChildType());
351     decIndentation();
352     printfln("</TypeList>");
353 }
354
355 private void printTypeError(TypeError t) {
356     printfln("<TypeError></TypeError>");
357 }
358
359 private void printTypeBool(TypeBool t) {
360     printfln("<TypeBool></TypeBool>");
361 }
362
363 @Override
364 public void visit(ASTDFEdge n) {
365     printfln("<Edge outputName=\"%s\" outputComponent=\"%s\" inputName=\"%s\" inputComponent =\"%s\">",
366             n.outputName, n.outputComponent.getId(), n.inputName, n.inputComponent.getId());
367     incIndentation();
368
369     printfln("<Connections>");
370     incIndentation();
371     for (TypeConnection.SingleConnection c : n.typeConnection.getConnections()) {
372         printfln("<Connection outputIndex=\"%d\" inputIndex = \"%d\"></Connection>",
373                 c.getOutputIndex(), c.getInputIndex());
374     }
375     decIndentation();
376     printfln("</Connections>");
377
378     decIndentation();
379     printfln("</Edge>");
380 }
381
382 @Override
383 public void visit(ASTDFComponentInput n) {
384     printfln("<InputComponent id=\"%s\" name=\"%s\" x=\"%d\" y=\"%d\" seq=\"%d\">",
385             n.inputId, n.inputTitle, n.componentLocation.x, n.componentLocation.y, n.
386             sequenceNumber);
386     incIndentation();
387
388     printfln("<InputOriginalType>");
389     incIndentation();
390     printType(n.inputOriginalType);
391     decIndentation();
392     printfln("</InputOriginalType>");
393
394     printfln("<InputType>");
395     incIndentation();
396     printType(n.inputType);
397     decIndentation();
398     printfln("</InputType>");
399
400     decIndentation();
401     printfln("</InputComponent>");
```

```
402     }
403
404     @Override
405     public void visit(ASTDFComponentOutput n) {
406         printfln("<OutputComponent id =\"%s\" name=\"%s\" x=\"%d\" y=\"%d\" seq=\"%d\">",
407                 n.outputId, n.outputTitle, n.componentLocation.x, n.componentLocation.y, n.
408                 sequenceNumber);
409         incIndentation();
410
411         printfln("<OutputOriginalType>");
412         incIndentation();
413         printType(n.outputOriginalType);
414         decIndentation();
415         printfln("</OutputOriginalType>");
416
417         printfln("<OutputType>");
418         incIndentation();
419         printType(n.outputType);
420         decIndentation();
421         printfln("</OutputType>");
422
423         decIndentation();
424         printfln("</OutputComponent>");
425
426     @Override
427     public void visit(ASTDF n) {
428         printfln("<DataFlow id=\"%s\">", n.dataFlowTitle);
429         incIndentation();
430
431         printfln("<Components>");
432         incIndentation();
433         for (ASTDFComponent c : n.dataFlowComponents) {
434             c.accept(this);
435         }
436         decIndentation();
437         printfln("</Components>");
438
439         printfln("<Edges>");
440         incIndentation();
441         for (ASTDFEdge e : n.dataFlowEdges) {
442             e.accept(this);
443         }
444         decIndentation();
445         printfln("</Edges>");
446
447         decIndentation();
448         printfln("</DataFlow>");
449     }
450
451     @Override
452     public void visit(ASTDFComponentArithmetic n) {
453         printfln("<ArithmeticComponent id=\"%s\" type=\"%s\" x=\"%d\" y=\"%d\" seq=\"%d\"></
ArithmeticComponent>",
454
```

```
454             n.arithmeticId, n.arithmeticType, n.componentLocation.x, n.componentLocation.y, n.  
455             sequenceNumber);  
456     }  
457  
458     @Override  
459     public void visit(ASTDFComponentStringOperation n) {  
460         printfln("<StringOperationComponent id=\"%s\" type=\"%s\" x=\"%d\" y=\"%d\" seq=\"%d\"></  
461             StringOperationComponent>",  
462             n.stringOperationId, n.stringOperationType, n.componentLocation.x, n.  
463             componentLocation.y, n.sequenceNumber);  
464     }  
465  
466     @Override  
467     public void visit(ASTDFComponentCast n) {  
468         printfln("<CastComponent id=\"%s\" x=\"%d\" y=\"%d\" seq=\"%d\">,  
469             n.castId, n.componentLocation.x, n.componentLocation.y, n.sequenceNumber);  
470         incIndentation();  
471  
472         printfln("<OriginalCastFrom>");  
473         incIndentation();  
474         printType(n.originalCastFrom);  
475         decIndentation();  
476         printfln("</OriginalCastFrom>");  
477  
478         printfln("<CastFrom>");  
479         incIndentation();  
480         printType(n.castFrom);  
481         decIndentation();  
482         printfln("</CastFrom>");  
483  
484         printfln("<CastTo>");  
485         incIndentation();  
486         printType(n.castTo);  
487         decIndentation();  
488         printfln("</CastTo>");  
489     }  
490  
491     @Override  
492     public void visit(ASTDFComponentComparison n) {  
493         printfln("<ComparisonComponent id=\"%s\" type=\"%s\" x=\"%d\" y=\"%d\" seq=\"%d\">,  
494             n.comparisonId, n.comparisonType, n.componentLocation.x, n.componentLocation.y, n.  
495             sequenceNumber);  
496         incIndentation();  
497  
498         printfln("<ComparisonDataType>");  
499         incIndentation();  
500         printType(n.comparisonDataType);  
501         decIndentation();  
502         printfln("</ComparisonDataType>");  
503  
504         printfln("<ComparisonOriginalDataType>");  
505         incIndentation();
```

```
505     printType(n.comparisonOriginalDataType);
506     decIndentation();
507     printfln("</ComparisonOriginalDataType>");
508
509     decIndentation();
510     printfln("</ComparisonComponent>");
511 }
512
513 @Override
514 public void visit(ASTDFComponentConstant n) {
515     printfln("<ConstantComponent id=\"%s\" x=\"%d\" y=\"%d\" seq=\"%d\">",
516             n.constantId, n.componentLocation.x, n.componentLocation.y, n.sequenceNumber);
517     incIndentation();
518     printConstant(n.constantValue);
519     decIndentation();
520     printfln("</ConstantComponent>");
521 }
522
523 @Override
524 public void visit(ASTDFComponentStringFormat n) {
525     String fmt1 = StringEscapeUtils.escapeXml11(n.stringFormat);
526     String fmt = fmt1.replaceAll("\n", "\r\n");
527     printfln("<StringFormatComponent id=\"%s\" fmt=\"%s\" x=\"%d\" y=\"%d\" seq=\"%d\"></
528 StringFormatComponent>",
529             n.stringFormatId, fmt, n.componentLocation.x, n.componentLocation.y, n.
530             sequenceNumber);
531 }
532
533 @Override
534 public void visit(ASTDFComponentRecordConstructor n) {
535     printfln("<RecordConstructorComponent id=\"%s\" x=\"%d\" y=\"%d\" seq=\"%d\">",
536             n.recordId, n.componentLocation.x, n.componentLocation.y, n.sequenceNumber);
537     incIndentation();
538     printType(n.recordType);
539     decIndentation();
540     printfln("</RecordConstructorComponent>");
541 }
542
543 @Override
544 public void visit(ASTDFComponentListOperation n) {
545     printfln("<ListOperationComponent id=\"%s\" type=\"%s\" x=\"%d\" y=\"%d\" seq=\"%d\">",
546             n.listOperationId, n.listOperationType, n.componentLocation.x, n.componentLocation
547             .y, n.sequenceNumber);
548     incIndentation();
549
550     for (Type t : n.listOperationOriginalTypes) {
551         printfln("<ListOperationOriginalType>");
552         incIndentation();
553         printType(t);
554         decIndentation();
555         printfln("</ListOperationOriginalType>");
556     }
557
558     for (Type t : n.listOperationTypes) {
559         printfln("<ListOperationType>");
```

```
557         incIndentation();
558         printType(t);
559         decIndentation();
560         printfln("</ListOperationType>");
561     }
562 
563     decIndentation();
564     printfln("</ListOperationComponent>");
565 }
566 
567 @Override
568 public void visit(ASTDFComponentDF n) {
569     printfln("<DataFlowComponent id=\"%s\" x=\"%d\" y=\"%d\" seq=\"%d\">",
570             n.getId(), n.componentLocation.x, n.componentLocation.y, n.sequenceNumber);
571     incIndentation();
572     n.dataFlow.accept(this);
573     decIndentation();
574     printfln("</DataFlowComponent>");
575 }
576 
577 @Override
578 public void visit(ASTDFComponentFunctional n) {
579     printfln("<FunctionalComponent id=\"%s\" subId=\"%s\" type=\"%s\" x=\"%d\" y=\"%d\" seq"
580             =" "%d\">",
581             n.functionalId, n.dataFlow.dataFlowTitle, n.functionalType, n.componentLocation.x,
582             n.componentLocation.y, n.sequenceNumber);
583     incIndentation();
584 
585     printfln("<FunctionalInputs>");
586     incIndentation();
587     for (RecordColumn c : n.subDataFlowInputs) {
588         printfln("<FunctionalInputColumn id=\"%s\">", c.getId());
589         incIndentation();
590         printType(c.getType());
591         decIndentation();
592         printfln("</FunctionalInputColumn>");
593     }
594     decIndentation();
595     printfln("<FunctionalOutputs>");
596     incIndentation();
597     for (RecordColumn c : n.subDataFlowOutputs) {
598         printfln("<FunctionalOutputColumn id=\"%s\">", c.getId());
599         incIndentation();
600         printType(c.getType());
601         decIndentation();
602         printfln("</FunctionalOutputColumn>");
603     }
604     decIndentation();
605     printfln("</FunctionalOutputs>");
606 
607     printfln("<FunctionalSubDataFlow>");
608     incIndentation();
609     n.dataFlow.accept(this);
```

```

610     decIndentation();
611     printfln("</FunctionalSubDataFlow>");
612 
613     decIndentation();
614     printfln("</FunctionalComponent>");
615 }
616 
617 @Override
618 public void visit(ASTDFComponentSink n) {
619     printfln("<SinkComponent id=\"%s\" x=\"%d\" y=\"%d\" seq=\"%d\">",
620             n.sinkId, n.componentLocation.x, n.componentLocation.y, n.sequenceNumber);
621     incIndentation();
622     printType(n.sinkType);
623     decIndentation();
624     printfln("</SinkComponent>");
625 }
626 }
```

LISTING E.29: lang/ast/ASTVisitorPrint.java

```

1 package lang.ast;
2 
3 import java.awt.Point;
4 import java.util.List;
5 
6 import lang.type.Type;
7 
8 public class ASTDFComponentCast extends ASTDFComponent {
9     String castId;
10    Type originalCastFrom;
11    Type castFrom;
12    Type castTo;
13 
14    public ASTDFComponentCast(Point location, Type origFrom, Type from, Type to,
15        String title, List<String> inputs, List<String> outputs, int seqNumber) {
16        super(location, title, inputs, outputs, seqNumber);
17        castId = ASTUtil.nextUniqueIdentifier();
18        originalCastFrom = origFrom;
19        castFrom = from;
20        castTo = to;
21    }
22 
23    @Override
24    public void accept(ASTVisitor v) {
25        v.visit(this);
26    }
27 
28    @Override
29    String getId() {
30        return castId;
31    }
32 }
33 }
```

LISTING E.30: lang/ast/ASTDFComponentCast.java

```
1 package lang.ast;
2
3 import java.awt.Point;
4 import java.util.List;
5
6 import lang.constant.Constant;
7
8 public class ASTDFComponentConstant extends ASTDFComponent {
9     String constantId;
10    Constant constantValue;
11
12    public ASTDFComponentConstant(Point location, Constant value, String title,
13                                    List<String> inputs, List<String> outputs, int seqNumber) {
14        super(location, title, inputs, outputs, seqNumber);
15        constantValue = value;
16        constantId = ASTUtil.nextUniqueIdentifier();
17    }
18
19    @Override
20    public void accept(ASTVisitor v) {
21        v.visit(this);
22    }
23
24    @Override
25    String getId() {
26        return constantId;
27    }
28
29 }
```

LISTING E.31: lang/ast/ASTDFComponentConstant.java

```
1 package lang.ast;
2
3 public interface ASTVisitor {
4     public void visit(ASTCF n);
5     public void visit(ASTCFTypeConnection n);
6     public void visit(ASTCFComponentDF n);
7     public void visit(ASTCFComponentIf n);
8     public void visit(ASTCFComponentStop n);
9     public void visit(ASTCFComponentStart n);
10    public void visit(ASTCFEdge n);
11    public void visit(ASTDFEdge n);
12    public void visit(ASTDFComponentInput n);
13    public void visit(ASTDFComponentOutput n);
14    public void visit(ASTDF n);
15    public void visit(ASTDFComponentArithmetic n);
16    public void visit(ASTDFComponentCast n);
17    public void visit(ASTDFComponentComparison n);
18    public void visit(ASTDFComponentConstant n);
19    public void visit(ASTDFComponentListOperation n);
20    public void visit(ASTDFComponentDF n);
21    public void visit(ASTDFComponentFunctional n);
22    public void visit(ASTDFComponentSink n);
```

```

23     public void visit(ASTCFComponentInteraction n);
24     public void visit(ASTCFComponentFail n);
25     public void visit(ASTDFComponentStringOperation n);
26     public void visit(ASTDFComponentRecordConstructor n);
27     public void visit(ASTDFComponentStringFormat n);
28 }
```

LISTING E.32: lang/ast/ASTVisitor.java

```

1 package lang.ast;
2
3 import java.awt.Point;
4 import java.util.ArrayList;
5 import java.util.List;
6
7 import lang.type.RecordColumn;
8
9 public class ASTDFComponentFunctional extends ASTDFComponent {
10     String functionalType;
11     ASTDF dataFlow;
12     String functionalId;
13     List<RecordColumn> subDataFlowInputs;
14     List<RecordColumn> subDataFlowOutputs;
15
16     public ASTDFComponentFunctional(Point location, String functionalType, List<RecordColumn>
17         inputs,
18         List<RecordColumn> outputs, ASTDF df, String title,
19         List<String> inputNames, List<String> outputNames, int seqNumber) {
20         super(location, title, inputNames, outputNames, seqNumber);
21         functionalId = ASTUtil.nextUniqueIdentifier();
22         this.functionalType = functionalType;
23         dataFlow = df;
24         subDataFlowInputs = new ArrayList<>(inputs);
25         subDataFlowOutputs = new ArrayList<>(outputs);
26     }
27
28     @Override
29     public void accept(ASTVisitor v) {
30         v.visit(this);
31     }
32
33     @Override
34     String getId() {
35         return functionalId;
36     }
37 }
```

LISTING E.33: lang/ast/ASTDFComponentFunctional.java

```

1 package lang.ast;
2
3 import lang.type.TypeConnection;
4
```

```

5  public class ASTCFTypeConnection implements ASTNode {
6      ASTCFComponent outputComponent;
7      ASTCFComponent inputComponent;
8      String outputName;
9      String inputName;
10     TypeConnection typeConnection;
11
12     public ASTCFTypeConnection(ASTCFComponent outputComponent,
13         String outputName, ASTCFComponent inputComponent, String inputName,
14         TypeConnection typeConnection) {
15         this.inputComponent = inputComponent;
16         this.outputComponent = outputComponent;
17         this.outputName = outputName;
18         this.inputName = inputName;
19         this.typeConnection = typeConnection;
20     }
21
22     @Override
23     public void accept(ASTVisitor v) {
24         v.visit(this);
25     }
26 }
```

LISTING E.34: lang/ast/ASTCFTypeConnection.java

```

1  package lang.ast;
2
3  import java.util.ArrayList;
4  import java.util.HashSet;
5  import java.util.List;
6  import java.util.Set;
7  import java.util.Stack;
8
9  public class ASTVisitorCheck implements ASTVisitor {
10     private ArrayList<Problem> problems = new ArrayList<>();
11     private Stack<String> parentStack = new Stack<>();
12
13     public List<Problem> getProblems(ASTCF root) {
14         root.accept(this);
15         return new ArrayList<>(problems);
16     }
17
18     private String getPrefix() {
19         if (parentStack.size() == 1)
20             return "Controlflow "+parentStack.lastElement()+" - ";
21         else
22             return "Dataflow "+parentStack.lastElement()+" - ";
23     }
24
25     public class Problem {
26         private String message;
27         private List<String> path;
28         private int startSequenceNumber;
29         private int endSequenceNumber = -1;
30         private int startEventIndex = -1;
```

```
31
32     public Problem(String msg, int seq) {
33         startSequenceNumber = seq;
34         message = msg;
35         path = new ArrayList<>(parentStack);
36     }
37
38     public Problem(String msg, int start, int eventIndex, int end) {
39         this(msg, start);
40         endSequenceNumber = end;
41         startEventIndex = eventIndex;
42     }
43
44     public String getMessage() {
45         return message;
46     }
47
48     public int getStartEventIndex() {
49         return startEventIndex;
50     }
51
52     public int getStartSequenceNumber() {
53         return startSequenceNumber;
54     }
55
56     public int getEndSequenceNumber() {
57         return endSequenceNumber;
58     }
59
60     public List<String> getPath() {
61         return new ArrayList<>(path);
62     }
63
64     @Override
65     public String toString() {
66         return getMessage();
67     }
68 }
69
70     @Override
71     public void visit(ASTCF n) {
72         parentStack.push(n.controlFlowTitle);
73
74         HashSet<ASTCFComponent> unusedComps = new HashSet<>(n.components);
75         for (ASTCFEdge e : n.connections) {
76             unusedComps.remove(e.endComponent);
77         }
78         if (unusedComps.size() > 0) {
79             String msgStart = getPrefix();
80             for (ASTCFComponent c : n.components) {
81                 if (!unusedComps.contains(c) || c instanceof ASTCFComponentStart)
82                     continue;
83                 String msgEnd = "component "+c.getTitle()+" is unreachable";
84                 problems.add(new Problem(msgStart+msgEnd, c.sequenceNumber));
85             }
86         }
87     }
88 }
```

```
86      }
87      for (ASTCFComponent c : n.components) {
88          if (unusedComps.contains(c) && !(c instanceof ASTCFComponentStart))
89              continue;
90          Set<String> eventSet = new HashSet<>(c.componentEventMap.values());
91          for (ASTCFEdge e : n.connections) {
92              if (!e.startComponent.equals(c))
93                  continue;
94              String event = c.componentEventMap.get(e.startEventIndex);
95              assert event != null;
96              eventSet.remove(event);
97          }
98          if (eventSet.size() == 0)
99              continue;
100         String msgStart = getPrefix();
101         String msgEnd = null;
102         if (eventSet.size() == 1) {
103             for (String evt : eventSet) {
104                 if (evt.equals("")) {
105                     msgEnd = "component '"+c.getTitle()+"' has unhandled event";
106                 } else {
107                     msgEnd = "component '"+c.getTitle()+"' has unhandled '"+evt+"' event";
108                 }
109             }
110         } else {
111             msgEnd = "component '"+c.getTitle()+"' has unhandled events";
112         }
113         problems.add(new Problem(msgStart+msgEnd, c.sequenceNumber));
114     }
115
116     for (ASTCFEdge e : n.connections) {
117         e.accept(this);
118     }
119     for (ASTCFComponent c : n.components) {
120         c.accept(this);
121     }
122
123     parentStack.pop();
124 }
125
126 private String getInvalidTypeConnectionMessage(String outputComp, String outputEvent, String
127 inputComp) {
128     if (outputEvent.equals("")) {
129         return getPrefix()+"branch from '"+outputComp+"'
130             " to '"+inputComp+"' has incomplete type connection";
131     }
132     return getPrefix()+"branch from '"+outputComp+"' event '"+outputEvent+
133         " to '"+inputComp+"' has incomplete type connection";
134 }
135
136 @Override
137 public void visit(ASTCFTypeConnection n) {}
138
139 @Override
140 public void visit(ASTCFComponentInteraction n) {}
```

```
140
141     @Override
142     public void visit(ASTCFComponentDF n) {
143         n.dataFlow.accept(this);
144     }
145
146     @Override
147     public void visit(ASTCFComponentIf n) {}
148
149     @Override
150     public void visit(ASTCFComponentStop n) {}
151
152     @Override
153     public void visit(ASTCFComponentFail n) {}
154
155     @Override
156     public void visit(ASTCFComponentStart n) {}
157
158     @Override
159     public void visit(ASTCFEdge n) {
160         outer: {
161             for (ASTCFTypeConnection c : n.typeConnections) {
162                 if (!c.typeConnection.isConnected())
163                     break outer;
164             }
165             if (n.typeConnections.size() < n.endComponent.componentInputs.size()) {
166                 break outer;
167             }
168             return;
169         }
170         String outputEvent = n.startComponent.componentEventMap.get(n.startEventIndex);
171         String msg = getInvalidTypeConnectionMessage(n.startComponent.getTitle(), outputEvent,
172             n.endComponent.getTitle());
173         problems.add(new Problem(msg, n.startComponent.sequenceNumber,
174             n.startEventIndex, n.endComponent.sequenceNumber));
175     }
176
177     @Override
178     public void visit(ASTDFEdge n) {}
179
180     @Override
181     public void visit(ASTDFComponentInput n) {}
182
183     @Override
184     public void visit(ASTDFComponentOutput n) {}
185
186     @Override
187     public void visit(ASTDF n) {
188         parentStack.push(n.dataFlowTitle);
189         for (ASTDFComponent c : n.dataFlowComponents) {
190             Set<String> unusedInputs = new HashSet<>(c.componentInputs);
191             Set<String> unusedOutputs = new HashSet<>(c.componentOutputs);
192             for (ASTDFEdge e : n.dataFlowEdges) {
193                 if (e.inputComponent.equals(c)) {
194                     unusedInputs.remove(e.inputName);
```

```
195         } else if (e.outputComponent.equals(c)) {
196             unusedOutputs.remove(e.outputName);
197         }
198     }
199     String prefix = getPrefix()+"component "+c.componentRefName+" ";
200     if (unusedInputs.size() > 0) {
201         String end = null;
202         if (unusedInputs.size() == 1) {
203             for (String s : unusedInputs)
204                 if (s.equals(""))
205                     end = "input is not connected";
206                 else
207                     end = "input "+s+" is not connected";
208         } else {
209             end = "has unconnected inputs";
210         }
211         problems.add(new Problem(prefix+end, c.sequenceNumber));
212     }
213     if (unusedOutputs.size() > 0) {
214         String end = null;
215         if (unusedOutputs.size() == 1) {
216             for (String s : unusedOutputs) {
217                 if (s.equals(""))
218                     end = "output is not connected";
219                 else
220                     end = "output "+s+" is not connected";
221             }
222         } else {
223             end = "has unconnected outputs";
224         }
225         problems.add(new Problem(prefix+end, c.sequenceNumber));
226     }
227 }
228
229     for (ASTDFEdge e : n.dataFlowEdges) {
230         e.accept(this);
231     }
232     for (ASTDFComponent c : n.dataFlowComponents) {
233         c.accept(this);
234     }
235
236     parentStack.pop();
237 }
238
239 @Override
240 public void visit(ASTDFComponentArithmetic n) {}
241
242 @Override
243 public void visit(ASTDFComponentStringOperation n) {}
244
245 @Override
246 public void visit(ASTDFComponentCast n) {}
247
248 @Override
249 public void visit(ASTDFComponentComparison n) {}
```

```
250
251     @Override
252     public void visit(ASTDFComponentRecordConstructor n) {}
253
254     @Override
255     public void visit(ASTDFComponentConstant n) {}
256
257     @Override
258     public void visit(ASTDFComponentStringFormat n) {}
259
260     @Override
261     public void visit(ASTDFComponentListOperation n) {}
262
263     @Override
264     public void visit(ASTDFComponentDF n) {
265         n.dataFlow.accept(this);
266     }
267
268     @Override
269     public void visit(ASTDFComponentFunctional n) {
270         n.dataFlow.accept(this);
271     }
272
273     @Override
274     public void visit(ASTDFComponentSink n) {}
275 }
```

LISTING E.35: lang/ast/ASTVisitorCheck.java

```
1 package lang.ast;
2
3 import java.awt.Point;
4 import java.util.ArrayList;
5 import java.util.List;
6
7 public abstract class ASTDFComponent implements ASTNode {
8     Point componentLocation;
9     int sequenceNumber;
10    String componentRefName;
11    List<String> componentInputs;
12    List<String> componentOutputs;
13
14    public ASTDFComponent(Point location, String title, List<String> inputs, List<String> outputs,
15                           int seqNumber) {
16        componentLocation = new Point(location);
17        sequenceNumber = seqNumber;
18        componentInputs = new ArrayList<>(inputs);
19        componentOutputs = new ArrayList<>(outputs);
20        componentRefName = title;
21    }
22
23    abstract String getId();
24
25    @Override
26    public boolean equals(Object obj) {
```

```

26     if (!(obj instanceof ASTDFComponent))
27         return false;
28     return ((ASTDFComponent)obj).getId().equals(getId());
29 }
30
31 @Override
32 public int hashCode() {
33     return getId().hashCode();
34 }
35 }
```

LISTING E.36: lang/ast/ASTDFComponent.java

```

1 package lang.ast;
2
3 import java.awt.Point;
4 import java.util.List;
5
6 import lang.type.Type;
7
8 public class ASTDFComponentInput extends ASTDFComponent {
9     String inputId;
10    Type inputOriginalType;
11    Type inputType;
12    String inputTitle;
13
14    public ASTDFComponentInput(Point location, Type origType, Type type, String name,
15        List<String> inputs, List<String> outputs, int seqNumber) {
16        super(location, name, inputs, outputs, seqNumber);
17        inputId = ASTUtil.nextUniqueIdentifier();
18        inputOriginalType = origType;
19        inputType = type;
20        inputTitle = name;
21    }
22
23    @Override
24    public void accept(ASTVisitor v) {
25        v.visit(this);
26    }
27
28    @Override
29    String getId() {
30        return inputId;
31    }
32 }
```

LISTING E.37: lang/ast/ASTDFComponentInput.java

```

1 package lang.ast;
2
3 public interface ASTNode {
4     void accept(ASTVisitor v);
5 }
```

LISTING E.38: lang/ast/ASTNode.java

```
1 package lang.ast;
2
3 import java.awt.Point;
4 import java.util.List;
5
6 import lang.type.Type;
7
8 public class ASTDFComponentComparison extends ASTDFComponent {
9     String comparisonId;
10    String comparisonType;
11    Type comparisonDataType;
12    Type comparisonOriginalDataType;
13
14    public ASTDFComponentComparison(Point location, String comparisonType, Type dataType, Type
15        origType,
16        String title, List<String> inputs, List<String> outputs, int seqNumber) {
17        super(location, title, inputs, outputs, seqNumber);
18        comparisonId = ASTUtil.nextUniqueIdentifier();
19        comparisonDataType = dataType;
20        this.comparisonType = comparisonType;
21        this.comparisonOriginalDataType = origType;
22    }
23
24    @Override
25    public void accept(ASTVisitor v) {
26        v.visit(this);
27    }
28
29    @Override
30    String getId() {
31        return comparisonId;
32    }
33 }
```

LISTING E.39: lang/ast/ASTDFComponentComparison.java

```
1 package lang.ast;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class ASTCFEdge implements ASTNode {
7     int startEventIndex;
8     ASTCFComponent startComponent;
9     ASTCFComponent endComponent;
10    List<ASTCFTypeConnection> typeConnections;
11
12    public ASTCFEdge(int eventIndex, ASTCFComponent start, ASTCFComponent end,
13        List<ASTCFTypeConnection> typeConns) {
```

```

14     startEventIndex = eventIndex;
15     startComponent = start;
16     endComponent = end;
17     typeConnections = new ArrayList<>(typeConns);
18 }
19
20 @Override
21 public void accept(ASTVisitor v) {
22     v.visit(this);
23 }
24 }
```

LISTING E.40: lang/ast/ASTCFEdge.java

```

1 package lang.ast;
2
3 import java.awt.Point;
4 import java.util.List;
5 import java.util.Map;
6
7 import lang.type.RecordColumn;
8
9 public class ASTCFComponentFail extends ASTCFComponent {
10     String failIdentifier;
11
12     public ASTCFComponentFail(Point location, List<RecordColumn> inputs, Map<Integer, String>
13         eventMap, int seqNum) {
14         super(location, inputs, eventMap, seqNum);
15         failIdentifier = ASTUtil.nextUniqueIdentifier();
16     }
17
18     @Override
19     public void accept(ASTVisitor v) {
20         v.visit(this);
21     }
22
23     @Override
24     public String getId() {
25         return failIdentifier;
26     }
27
28     @Override
29     public String getTitle() {
30         return "Fail";
31     }
32 }
```

LISTING E.41: lang/ast/ASTCFComponentFail.java

```

1 package lang.ast;
2
3 import java.awt.Point;
4 import java.util.List;
5
```

```

6  public class ASTDFComponentDF extends ASTDFComponent {
7      ASTDF dataFlow;
8
9      public ASTDFComponentDF(Point location, ASTDF df,
10          List<String> inputs, List<String> outputs, int seqNumber) {
11          super(location, df.dataFlowTitle, inputs, outputs, seqNumber);
12          dataFlow = df;
13      }
14
15      @Override
16      public void accept(ASTVisitor v) {
17          v.visit(this);
18      }
19
20      @Override
21      String getId() {
22          return dataFlow.dataFlowTitle;
23      }
24
25 }
```

LISTING E.42: lang/ast/ASTDFComponentDF.java

```

1 package lang.ast;
2
3 import lang.type.TypeConnection;
4
5 public class ASTDFEdge implements ASTNode {
6     String outputName;
7     String inputName;
8     ASTDFComponent outputComponent;
9     ASTDFComponent inputComponent;
10    TypeConnection typeConnection;
11
12    public ASTDFEdge(String outputName, ASTDFComponent output,
13        String inputName, ASTDFComponent input, TypeConnection conn) {
14        this.outputName = outputName;
15        this.inputName = inputName;
16        outputComponent = output;
17        inputComponent = input;
18        typeConnection = conn;
19    }
20
21    @Override
22    public void accept(ASTVisitor v) {
23        v.visit(this);
24    }
25 }
```

LISTING E.43: lang/ast/ASTDFEdge.java

```

1 package lang.ast;
2
3 import java.awt.Point;
```

```

4 import java.util.List;
5
6 public class ASTDFComponentStringOperation extends ASTDFComponent {
7     String stringOperationId;
8     String stringOperationType;
9
10    public ASTDFComponentStringOperation(Point location, String type, String title,
11                                         List<String> inputs, List<String> outputs, int seqNumber) {
12        super(location, title, inputs, outputs, seqNumber);
13        stringOperationId = ASTUtil.nextUniqueIdentifier();
14        stringOperationType = type;
15    }
16
17    @Override
18    public void accept(ASTVisitor v) {
19        v.visit(this);
20    }
21
22    @Override
23    String getId() {
24        return stringOperationId;
25    }
26
27 }
```

LISTING E.44: lang/ast/ASTDFComponentStringOperation.java

```

1 package lang.ast;
2
3 import java.awt.Point;
4 import java.util.List;
5 import java.util.Map;
6
7 import lang.type.RecordColumn;
8
9 public class ASTCFComponentStart extends ASTCFComponent {
10    String startIdentifier;
11
12    public ASTCFComponentStart(Point location, List<RecordColumn> inputs, Map<Integer, String>
13                               eventMap, int seqNum) {
14        super(location, inputs, eventMap, seqNum);
15        startIdentifier = ASTUtil.nextUniqueIdentifier();
16    }
17
18    @Override
19    public void accept(ASTVisitor v) {
20        v.visit(this);
21    }
22
23    @Override
24    String getId() {
25        return startIdentifier;
26    }
27    @Override
```

```
28     public String getTitle() {
29         return "Start";
30     }
31 }
```

LISTING E.45: lang/ast/ASTCFComponentStart.java

```
1 package lang.ast;
2
3 import java.awt.Point;
4 import java.util.ArrayList;
5 import java.util.Arrays;
6 import java.util.List;
7 import java.util.Map;
8
9 import lang.type.RecordColumn;
10
11 public class ASTCFComponentDF extends ASTCFComponent {
12     List<String> normalDataFlowEvents;
13     boolean dataFlowCanFail;
14     ASTDF dataFlow;
15
16     public ASTCFComponentDF(String title, Point location, List<RecordColumn> inputs,
17                             List<RecordColumn> outputs, List<ASTDFComponent> components, Map<Integer, String>
18                             eventMap, int seqNum, boolean canFail) {
19         super(location, inputs, eventMap, seqNum);
20         normalDataFlowEvents = new ArrayList<>(Arrays.asList(""));
21         dataFlow = new ASTDF(title, inputs, outputs, false);
22         dataFlow.addAllComponents(components);
23         dataFlowCanFail = canFail;
24     }
25
26     @Override
27     public void accept(ASTVisitor v) {
28         v.visit(this);
29     }
30
31     @Override
32     String getId() {
33         return dataFlow.dataFlowTitle;
34     }
35
36     public void addDataFlowEdge(ASTDFEdge edge) {
37         dataFlow.dataFlowEdges.add(edge);
38     }
39
40     public void addAllDataFlowEdges(List<ASTDFEdge> edges) {
41         dataFlow.dataFlowEdges.addAll(edges);
42     }
43
44     @Override
45     public String getTitle() {
46         return dataFlow.dataFlowTitle;
47     }
}
```

LISTING E.46: lang/ast/ASTCFComponentDF.java

```
1 package lang.ast;
2
3 import java.awt.Point;
4 import java.util.List;
5
6 import lang.type.Type;
7
8 public class ASTDFComponentRecordConstructor extends ASTDFComponent {
9     String recordId;
10    Type recordType;
11
12    public ASTDFComponentRecordConstructor(Point location, Type type, String title,
13        List<String> inputs, List<String> outputs, int seqNumber) {
14        super(location, title, inputs, outputs, seqNumber);
15        recordId = ASTUtil.nextUniqueIdentifier();
16        recordType = type;
17    }
18
19    @Override
20    public void accept(ASTVisitor v) {
21        v.visit(this);
22    }
23
24    @Override
25    String getId() {
26        return recordId;
27    }
28
29 }
```

LISTING E.47: lang/ast/ASTDFComponentRecordConstructor.java

```
1 package lang.ast;
2
3 import java.awt.Point;
4 import java.util.ArrayList;
5 import java.util.HashMap;
6 import java.util.List;
7 import java.util.Map;
8
9 import lang.type.RecordColumn;
10
11 public abstract class ASTCFComponent implements ASTNode {
12     Point componentLocation;
13     int sequenceNumber;
14     List<RecordColumn> componentInputs;
15     Map<Integer, String> componentEventMap;
16
17     public ASTCFComponent(Point location, List<RecordColumn> inputs, Map<Integer, String> eventMap,
18         int seqNum) {
```

```

18     componentLocation = new Point(location);
19     sequenceNumber = seqNum;
20     componentInputs = new ArrayList<>(inputs);
21     componentEventMap = new HashMap<>(eventMap);
22 }
23
24 abstract String getId();
25 abstract String getTitle();
26
27 @Override
28 public boolean equals(Object obj) {
29     if (!(obj instanceof ASTCFComponent))
30         return false;
31     return ((ASTCFComponent)obj).getId().equals(getId());
32 }
33
34 @Override
35 public int hashCode() {
36     return getId().hashCode();
37 }
38 }
```

LISTING E.48: lang/ast/ASTCFComponent.java

```

1 package lang.ast;
2
3 public class ASTUtil {
4     private static Integer nextId = new Integer(0);
5
6     public static String nextUniqueIdentifier() {
7         String ret = "("+nextId+ ")";
8         nextId++;
9         return ret;
10    }
11 }
```

LISTING E.49: lang/ast/ASTUtil.java

```

1 package lang.ast;
2
3 import java.awt.Point;
4 import java.util.List;
5 import java.util.Map;
6
7 import lang.type.RecordColumn;
8
9 public class ASTCFComponentStop extends ASTCFComponent {
10     String stopIdentifier;
11
12     public ASTCFComponentStop(Point location, List<RecordColumn> inputs, Map<Integer,String>
13     eventMap, int seqNum) {
14         super(location, inputs, eventMap, seqNum);
15         stopIdentifier = ASTUtil.nextUniqueIdentifier();
16     }
17 }
```

```
16
17     @Override
18     public void accept(ASTVisitor v) {
19         v.visit(this);
20     }
21
22     @Override
23     public String getId() {
24         return stopIdentifier;
25     }
26
27     @Override
28     public String getTitle() {
29         return "Stop";
30     }
31 }
```

LISTING E.50: lang/ast/ASTCFComponentStop.java

```
1 package lang.ast;
2
3 import java.io.ByteArrayOutputStream;
4 import java.io.File;
5 import java.io.FileNotFoundException;
6 import java.io.FileOutputStream;
7 import java.io.IOException;
8 import java.io.PrintStream;
9 import java.io.PrintWriter;
10 import java.util.ArrayList;
11 import java.util.Collections;
12 import java.util.HashMap;
13 import java.util.HashSet;
14 import java.util.LinkedList;
15 import java.util.List;
16 import java.util.Map;
17 import java.util.Set;
18 import java.util.Stack;
19
20 import org.apache.commons.lang3.StringEscapeUtils;
21 import org.apache.http.HttpResponse;
22 import org.apache.http.client.HttpClient;
23 import org.apache.http.client.methods.HttpPost;
24 import org.apache.http.entity.StringEntity;
25 import org.apache.http.impl.client.HttpClients;
26 import org.apache.http.util.EntityUtils;
27
28 import gui.controlflow.ControlPart;
29 import gui.dataflow.DataPartArithmetic;
30 import gui.dataflow.DataPartComparison;
31 import gui.dataflow.DataPartFunctional;
32 import gui.dataflow.DataPartListOperation;
33 import gui.dataflow.DataPartStringOperation;
34 import lang.constant.Constant;
35 import lang.constant.ConstantBool;
36 import lang.constant.ConstantList;
```

```
37 import lang.constant.ConstantNumber;
38 import lang.constant.ConstantRecord;
39 import lang.constant.ConstantString;
40 import lang.constant.RecordValue;
41 import lang.type.RecordColumn;
42 import lang.type.Type;
43 import lang.type.TypeAuto;
44 import lang.type.TypeCode;
45 import lang.type.TypeConnection;
46 import lang.type.TypeList;
47 import lang.type.TypeRecord;
48
49 public class ASTVisitorGen implements ASTVisitor {
50     private final static int indentationIncrement = 2;
51     private final static Map<String, String> emptyStringStringMap = new HashMap<>();
52
53     private static int nextConnectionNodeId = 0;
54     private static String getNextConnectionNodeName() {
55         return genDataResultId(nextConnectionNodeId++);
56     }
57     private static void resetConnectionNodeId() {
58         nextConnectionNodeId = 0;
59     }
60
61     public static void compile(ASTCF root) throws IOException {
62         ASTVisitorGen v = new ASTVisitorGen();
63         root.accept(v);
64
65         HttpClient httpclient = HttpClients.createDefault();
66         HttpPost httppost = new HttpPost("http://localhost:5005/Hs/" + v.moduleName + "." + v.platform +
67             ".+v.userGroup");
68         httppost.setEntity(new StringEntity(v.outputStream.toString("UTF-8")));
69         HttpResponse resp = httpclient.execute(httppost);
70
71         String msg = EntityUtils.toString(resp.getEntity());
72         if (!msg.startsWith("ok"))
73             throw new IOException(msg);
74     }
75
76     private final static String leftParenId = "L",
77         rightParenId = "R",
78         nameSepId = "_0",
79         dataResultId = "t",
80         lowerCaseStart = "_";
81
82     private class DataComponentNodeIO {
83         String name;
84         DataComponentNode node;
85         public DataComponentNodeIO(DataComponentNode node, String name) {
86             this.node = node;
87             this.name = name;
88         }
89         @Override
90         public boolean equals(Object obj) {
```

```
91         if (!(obj instanceof DataComponentNodeIO))
92             return false;
93         DataComponentNodeIO io = (DataComponentNodeIO)obj;
94         return io.node.equals(node) && io.name.equals(name);
95     }
96
97     @Override
98     public int hashCode() {
99         return node.hashCode() ^ name.hashCode();
100    }
101 }
102
103 private class DataComponentNode {
104     ASTDFComponent component;
105     HashMap<DataComponentNodeIO,ASTDFEdge> outputEdges = new HashMap<>();
106     HashMap<DataComponentNodeIO,ASTDFEdge> inputEdges = new HashMap<>();
107     HashMap<String,String> outputVars = new HashMap<>();
108
109     private int visitCount = 0;
110
111     public DataComponentNode(ASTDFComponent c) {
112         component = c;
113     }
114
115     public void start() {
116         component.accept(ASTVisitorGen.this);
117         for (DataComponentNodeIO s : outputEdges.keySet())
118             s.node.visit();
119     }
120
121     public void visit() {
122         ++visitCount;
123         if (visitCount == inputEdges.size()) {
124             component.accept(ASTVisitorGen.this);
125             for (DataComponentNodeIO s : outputEdges.keySet()) {
126                 s.node.visit();
127             }
128         }
129     }
130 }
131
132 private class DataComponentTree {
133     ArrayList<DataComponentNode> roots = new ArrayList<>();
134     Map<ASTDFComponent,DataComponentNode> nodeMap = new HashMap<>();
135
136     public void link(ASTDFEdge e) {
137         ASTDFComponent pred = e.outputComponent;
138         ASTDFComponent succ = e.inputComponent;
139         DataComponentNode predNode = nodeMap.get(pred);
140         if (predNode == null) {
141             predNode = new DataComponentNode(pred);
142             nodeMap.put(pred, predNode);
143         }
144         DataComponentNode succNode = nodeMap.get(succ);
145         if (succNode == null) {
```

```
146             succNode = new DataComponentNode(succ);
147             nodeMap.put(succ, succNode);
148         }
149         predNode.outputEdges.put(new DataComponentNodeIO(succNode, e.inputName), e);
150         succNode.inputEdges.put(new DataComponentNodeIO(predNode, e.inputName), e);
151     }
152
153     public void prepare() {
154         for (DataComponentNode n : nodeMap.values()) {
155             if (n.inputEdges.isEmpty())
156                 roots.add(n);
157         }
158     }
159 }
160
161 private static class ControlEvent {
162     ASTCFComponent startComponent;
163     String event;
164
165     public ControlEvent(ASTCFComponent c, String e) {
166         startComponent = c;
167         event = e;
168     }
169
170     @Override
171     public int hashCode() {
172         return event.hashCode() ^ startComponent.hashCode();
173     }
174
175     @Override
176     public boolean equals(Object obj) {
177         if (obj instanceof ControlEvent) {
178             ControlEvent other = (ControlEvent)obj;
179             return other.startComponent.equals(startComponent) && other.event.equals(event);
180         }
181         return false;
182     }
183 }
184
185 private static class ControlConnection {
186     ASTCFComponent inputComponent;
187     List<ASTCFTypeConnection> typeConnections;
188
189     public ControlConnection(ASTCFComponent c, List<ASTCFTypeConnection> t) {
190         inputComponent = c;
191         typeConnections = t;
192     }
193 }
194
195 private Map<ControlEvent,ControlConnection> controlFlowEventMap = new HashMap<>();
196
197 private PrintWriter output;
198 private ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
199 private String moduleName;
200 private String platform;
```

```
201     private String userGroup;
202     private String namePrefix = nameSepId;
203     private String currControlFunc;
204     private Set<String> existingControlFuncs = new HashSet<>();
205     private int startSequenceNumber;
206     private boolean inDataFlow = false;
207     private boolean inControlFlow = true;
208     private boolean isPreprocessing;
209     private Map<ASTCFComponent, String> componentErrorEventMap = new HashMap<>();
210     private DataComponentTree dataComponentTree;
211     private Map<String, String> dataOutputMap = new HashMap<>();
212     private int letCount;
213
214     private List<RecordColumn> sortColumns(List<RecordColumn> s) {
215         List<RecordColumn> ret = new ArrayList<>(s);
216         Collections.sort(ret, (x,y) ->
217             x.getId().compareTo(y.getId())
218         );
219         return ret;
220     }
221
222     private String asPatternStr(String s) {
223         return lowerCaseStart+s;
224     }
225
226     private String getColumnPattern(List<RecordColumn> columns) {
227         List<RecordColumn> cols = sortColumns(columns);
228         String arg = "(";
229         for (int i = 0; i < cols.size()-1; i++)
230             arg += lowerCaseStart+cols.get(i).getId() + ", ";
231         if (cols.size() > 0)
232             arg += lowerCaseStart+cols.get(cols.size()-1).getId();
233         arg += ")";
234         return arg;
235     }
236
237     private void printNest(int nest) {
238         for (int i = 0; i < nest; i++) {
239             for (int j = 0; j < indentationIncrement; j++)
240                 output.print(' ');
241         }
242     }
243
244     private void println(int nest, String s) {
245         if (!isPreprocessing) {
246             assert !s.contains("null");
247             printNest(nest);
248             output.println(s);
249         }
250     }
251
252     private boolean allNonNull(Object[] objs) {
253         for (Object o : objs)
254             if (o == null)
255                 return false;
```

```
256         return true;
257     }
258
259     private void printf(int nest, String fmt, Object... objs) {
260         if (!isPreprocessing) {
261             assert allNonNull(objs);
262             printNest(nest);
263             output.printf(fmt, objs);
264         }
265     }
266
267     private String asIdentifier(String s, boolean upperCaseStart) {
268         StringBuilder ret = new StringBuilder();
269         char[] cs = s.toCharArray();
270         if (upperCaseStart) {
271             cs[0] = Character.toUpperCase(cs[0]);
272         } else {
273             ret.append(lowerCaseStart);
274         }
275         for (int i = 0; i < cs.length; i++) {
276             if (cs[i] == '(') {
277                 if (upperCaseStart)
278                     ret.insert(0, 'L');
279                 else
280                     ret.insert(0, 'l');
281                 ret.append(leftParenId);
282             } else if (cs[i] == ')') {
283                 if (upperCaseStart)
284                     ret.insert(0, 'R');
285                 else
286                     ret.insert(0, 'r');
287                 ret.append(rightParenId);
288             } else {
289                 ret.append(cs[i]);
290             }
291         }
292
293         return ret.toString();
294     }
295
296     private String genFuncId(ASTNode node, String... more) {
297         String name;
298         if (node instanceof ASTCFComponentStart) {
299             return "start";
300         } else if (node instanceof ASTCFComponent) {
301             name = "c"+((ASTCFComponent)node).getId();
302         } else if (node instanceof ASTDF) {
303             name = "d"+((ASTDF)node).dataFlowTitle;
304         } else {
305             name = "d"+((ASTDFComponent)node).getId();
306         }
307         String ret = namePrefix+asIdentifier(name, false);
308         for (String s : more) {
309             ret+= nameSepId+asIdentifier(s, false);
310         }
311     }
```

```
311         return ret;
312     }
313
314     private enum ControlVarIdType {OUTPUT, INPUT};
315     private String genControlVarId(ControlVarIdType t, int seq, String name) {
316         if (t == ControlVarIdType.INPUT)
317             return "_" + seq + "_" + name + "_I";
318         if (seq == startSequenceNumber && name.equals("Username"))
319             return "_username";
320         return "_" + seq + "_" + name + "_Q";
321     }
322
323     private static String genDataResultId(int id) {
324         return dataResultId + id;
325     }
326
327     @Override
328     public void visit(ASTCF n) {
329         moduleName = new String(asIdentifier(n.controlFlowTitle, true));
330         platform = n.controlFlowPlatform;
331         if (n.controlFlowGroup.equals("Normal"))
332             userGroup = "0";
333         else
334             userGroup = "1";
335
336         output = new PrintWriter(outputStream);
337
338         try {
339             println(0, "{-# Language OverloadedStrings #-}\n");
340             println(0, "module " + moduleName + " where\n");
341             println(0, "import SeqServiceIO");
342             println(0, "import SeqActIO");
343             println(0, "import ServiceError");
344             println(0, "import Internal.Json");
345             println(0, "import Services");
346             println(0, "import Runtime");
347             println(0, "import Control.Monad.Trans.Either");
348             println(0, "import Data.Scientific");
349             println(0, "import qualified Data.HashMap.Strict as Map");
350             println(0, "import qualified Data.Text as Text");
351             println(0, "import Control.Monad.Trans.Class (lift)");
352             println(0, "import Text.Read (readMaybe)");
353
354             for (ASTCFEdge e : n.connections)
355                 e.accept(this);
356
357             isPreprocessing = true;
358             for (ASTCFComponent c : n.components) {
359                 if (c instanceof ASTCFComponentStart) {
360                     c.accept(this);
361                     break;
362                 }
363             }
364
365             currControlFunc = null;
```

```
366     existingControlFuncs.clear();
367     isPreprocessing = false;
368     for (ASTCFComponent c : n.components) {
369         if (c instanceof ASTCFComponentStart) {
370             c.accept(this);
371             break;
372         }
373     }
374
375     existingControlFuncs.clear();
376     currControlFunc = null;
377     inControlFlow = false;
378     for (ASTCFComponent c : n.components) {
379         c.accept(this);
380     }
381     } finally {
382         output.close();
383     }
384 }
385
386 private boolean startControlFunc(ASTCFComponent node) {
387     return startControlFunc(genFuncId(node));
388 }
389
390 private boolean startControlFunc(String id) {
391     letCount = 0;
392     if (existingControlFuncs.contains(id)) {
393         currControlFunc = null;
394         return false;
395     }
396     currControlFunc = id;
397     existingControlFuncs.add(id);
398     println(0, "\n"+id+" :: IO (Either ServiceError ())");
399     println(0, id+" = runEitherT $ do {");
400     return true;
401 }
402
403 private boolean startDataFunc(String id, List<RecordColumn> inputs,
404     List<RecordColumn> outputs, boolean fixedInputArgs) {
405     assert !inDataFlow;
406
407     letCount = 0;
408     if (existingControlFuncs.contains(id)) {
409         return false;
410     }
411
412     if (!fixedInputArgs)
413         inputs = sortColumns(inputs);
414
415     String args = "(";
416     String argType = "(";
417     for (int i = 0; i < inputs.size()-1; i++) {
418         argType += "Json,";
419         args += asPatternStr(inputs.get(i).getId()) + ", ";
420     }
```

```
421     if (!inputs.isEmpty()) {
422         argType += "Json";
423         args += asPatternStr(inputs.get(inputs.size()-1).getId());
424     }
425     argType += ")";
426     args += ")";
427
428     String retType = "(";
429     for (int i = 0; i < outputs.size()-1; i++) {
430         retType += "Json,";
431     }
432     if (!outputs.isEmpty()) {
433         retType += "Json";
434     }
435     retType += ")";
436
437     println(0, "\n"+id+" :: "+argType+" -> ActID -> SeqActIO "+retType);
438     println(0, id+" "+args+" actID = do {");
439
440     existingControlFuncs.add(id);
441     inDataFlow = true;
442     return true;
443 }
444
445 private void endControlFunc() {
446     println(1, "return ();");
447     String brackets = "}";
448     for (int i = 0; i < letCount; i++)
449         brackets += "}";
450     println(0, brackets);
451 }
452
453 private void endDataFunc(String ret) {
454     println(1, "return "+ret+";");
455     String brackets = "}";
456     for (int i = 0; i < letCount; i++)
457         brackets += "}";
458     println(0, brackets);
459     inDataFlow = false;
460 }
461
462 private void genInteraction(String func, String interaction,
463     Map<String, String> inputMap, Map<String, String> events,
464     Map<String, String> resultMap) {
465     String args = "";
466     ArrayList<String> input = new ArrayList<>(inputMap.keySet());
467     for (int i = 0; i < input.size()-1; i++) {
468         args += input.get(i) + "=" + inputMap.get(input.get(i)) + ", ";
469     }
470     if (input.size() > 0) {
471         String k = input.get(input.size()-1);
472         args += k + "=" + inputMap.get(k);
473     }
474
475     String res = "";
```

```
476     ArrayList<String> output = new ArrayList<>(resultMap.keySet());
477     for (int i = 0; i < output.size()-1; i++) {
478         res += resultMap.get(output.get(i))+"="+output.get(i)+", ";
479     }
480     if (output.size() > 0) {
481         String k = output.get(output.size()-1);
482         res += resultMap.get(k)+"="+k;
483     }
484
485     String evs = "";
486     ArrayList<String> eventAry = new ArrayList<>(events.keySet());
487     for (int i = 0; i < eventAry.size()-1; i++) {
488         evs += eventAry.get(i) + "="+ events.get(eventAry.get(i)) + ", ";
489     }
490     if (eventAry.size() > 0) {
491         String k = eventAry.get(eventAry.size()-1);
492         evs += k + "="+ events.get(k);
493     }
494
495     println(0, "{{-- Interact "+func+" :: "+interaction+"\n" +
496             " {"+args+"}\n" +
497             " {"+evs+"} ->\n" +
498             " {"+res+"}\n" +
499             "--}}");
500 }
501
502 @Override
503 public void visit(ASTCFTypeConnection n) {}
504
505 private void genSaveInDb(int nest, String pattern,
506                         ASTCFComponent n, List<RecordColumn> cs) {
507
508     for (RecordColumn col : cs) {
509         String s = genControlVarId(ControlVarIdType.OUTPUT, n.sequenceNumber, col.getId());
510         printf(nest, "lift (runSeqServiceIO (setVarSIO \'%s\' %s)) >> hoistEither;\n", s,
511                asPatternStr(col.getId()));
512     }
513 }
514
515 @Override
516 public void visit(ASTCFComponentDF n) {
517     if (!inControlFlow) {
518         n.dataFlow.accept(this);
519         return;
520     }
521
522     // Obtain function argument.
523     for (RecordColumn col : n.dataFlow.dataFlowInputs) {
524         String varId = genControlVarId(ControlVarIdType.INPUT,
525                                         n.sequenceNumber, col.getId());
526         printf(1, "%s <- lift (runSeqServiceIO (getVarSIO \'%s\')) >> hoistEither;\n",
527                asPatternStr(col.getId()), varId);
528     }
529     String arg = getColumnPattern(n.dataFlow.dataFlowInputs);
530     String res = getColumnPattern(n.dataFlow.dataFlowOutputs);
```

```
529     String evt = n.componentInstance.get(1);
530     if (!n.dataFlowCanFail) {
531         printf(1, "%s <- lift (runSeqActIO (%s %s)) >>= hoistEither;\n", res, genFuncId(n.
532             dataFlow), arg);
533         genSaveInDb(1, res, n, n.dataFlow.dataFlowOutputs);
534         ControlConnection nextControlConn = controlFlowEventMap.get(new ControlEvent(n, evt));
535         genControlArgs(1, nextControlConn);
536         nextControlConn.inputComponent.accept(this);
537         return;
538     } else {
539         String tmp0 = genDataResultId(0);
540         String tmp1 = genDataResultId(1);
541         printf(1, "%s <- lift (runSeqActIO (%s %s));\n", tmp0, genFuncId(n.dataFlow), arg);
542         printf(1, "case %s of {\n", tmp0);
543         printf(2, "Right "+res+" -> do {\n");
544         genSaveInDb(3, res, n, n.dataFlow.dataFlowOutputs);
545         println(3, "lift (runSeqServiceIO (setVarSIO \"cond\" (boolToJson True))) >>=
546             hoistEither;");
547         println(2, "};");
548         printf(2, "Left %s -> do {\n", tmp1);
549         printf(3, "lift (runSeqServiceIO (setVarSIO \"%s\" (strToJson (show %s)))) >>=
550             hoistEither;\n",
551             currControlFunc+"_Error", tmp1);
552         println(3, "lift (runSeqServiceIO (setVarSIO \"cond\" (boolToJson False))) >>=
553             hoistEither;");
554         println(2, "};");
555         println(1, "};");
556         endControlFunc();
557         Map<String, String> in = new HashMap<>();
558         in.put("cond", "cond");
559         Map<String, String> events = getEventMap(n);
560         Map<String, String> fake = new HashMap<>();
561         for (String k : events.keySet()) {
562             if (k.equals(ControlPart.errorEvent)) {
563                 fake.put("False", events.get(k));
564             } else {
565                 fake.put("True", events.get(k));
566             }
567         }
568     }
569
570     @Override
571     public void visit(ASTCFComponentIf n) {
572         if (!inControlFlow)
573             return;
574
575         endControlFunc();
576
577         RecordColumn in = n.componentInstance.get(0);
578         String varName = genControlVarId(ControlVarIdType.INPUT, n.sequenceNumber, in.getId());
579     }
```

```
580     Map<String, String> args = new HashMap<>();
581     args.put("cond", varName);
582     Map<String, String> evs = getEventMap(n);
583     genInteraction(currControlFunc, "Runtime.if", args, evs, emptyStringStringMap);
584     genContinuation(n);
585 }
586
587 private Map<String, String> getEventMap(ASTCFComponent n) {
588     HashMap<String, String> evs = new HashMap<>();
589     for (String e : new HashSet<>(n.componentEventMap.values())) {
590         if (e.equals(""))
591             e = lowerCaseStart;
592         evs.put(e, genFuncId(n, e));
593     }
594     return evs;
595 }
596
597 private void genContinuation(ASTCFComponent from) {
598     Set<String> evs = new HashSet<>(from.componentEventMap.values());
599
600     String funBefore = currControlFunc;
601     for (String e : evs) {
602         ControlConnection nextControlConn = controlFlowEventMap.get(new ControlEvent(from, e))
603         ;
604         if (e.equals(""))
605             e = lowerCaseStart;
606
607         String fun = genFuncId(from, e);
608         if (!startControlFunc(fun)) {
609             break;
610         }
611
612         genControlArgs(1, nextControlConn);
613         endControlFunc();
614         String nextFun = genFuncId(nextControlConn.inputComponent);
615         HashMap<String, String> events = new HashMap<>();
616         events.put("Event", nextFun);
617         genInteraction(fun, "Runtime.nop", emptyStringStringMap,
618                         events, emptyStringStringMap);
619
620         if (isPreprocessing && e.equals(ControlPart.errorEvent)) {
621             componentErrorEventMap.put(from, funBefore+_+ControlPart.errorEvent);
622         }
623         if (!startControlFunc(nextFun)) {
624             continue;
625         }
626         nextControlConn.inputComponent.accept(this);
627     }
628 }
629
630 private class ConnectionNode {
631     ConnectionNode parent;
632     TypeConnection.PathNode node;
633     ArrayList<ConnectionNode> children = new ArrayList<>();
```

```
634     String varName;
635     ArrayList<ConnectionNode> links = new ArrayList<>();
636
637     boolean isPrinted;
638
639     ConnectionNode(ConnectionNode parent, TypeConnection.PathNode node) {
640         this.parent = parent;
641         this.node = node;
642         this.varName = getNextConnectionNodeName();
643     }
644
645     ConnectionNode getNodeAfter(TypeConnection.PathNode node) {
646         ConnectionNode n = new ConnectionNode(this, node);
647         children.add(n);
648         return n;
649     }
650
651     void linkWith(ConnectionNode inputNode) {
652         if (links.contains(inputNode))
653             return;
654         links.add(inputNode);
655         inputNode.links.add(this);
656     }
657
658     void printDot(PrintStream out) {
659         if (isPrinted)
660             return;
661         isPrinted = true;
662
663         for (ConnectionNode c : children)
664             out.println(" "+varName+"_"+getNodeString()+" -> "+c.varName+"_"+c.getNodeString()
665             ()+";");
666         for (ConnectionNode c : children)
667             c.printDot(out);
668         for (ConnectionNode link : links) {
669             out.println(" "+varName+"_"+getNodeString()+" -> "+link.varName+"_"+link.
670             getNodeString()+" [style=dotted];");
671             if (isLeaf()) {
672                 link.printDotReversed(out);
673             }
674         }
675
676         void genSubTree(int nest) {
677             if (node.getType().getTypeCode() == TypeCode.LIST)
678                 return;
679
680             switch (node.getType().getTypeCode()) {
681             case BOOL:
682             case ERROR:
683             case NUMBER:
684             case STRING:
685                 return;
686             case AUTO:
```

```
687         for (ConnectionNode n : children) {
688             n.genSubTree(nest);
689             ++letCount;
690             println(nest, "let "+varName+" = "+n.varName+" in do {\n");
691         }
692         return;
693     }
694     case RECORD:
695         String tmp = getNextConnectionNodeName();
696         ++letCount;
697         printf(nest, "let %s = Map.empty :: Map.HashMap Text.Text Json in do {\n", tmp);
698         for (ConnectionNode n : children) {
699             n.genSubTree(nest);
700             String tmp2 = getNextConnectionNodeName();
701             ++letCount;
702             printf(nest, "let %s = Map.insert \'%s\' %s %s in do {\n", tmp2, n.node.
703             getColumnName(), n.varName, tmp);
704             tmp = tmp2;
705         }
706         ++letCount;
707         printf(nest, "let %s = mapToJson %s in do {\n", varName, tmp);
708         return;
709     }
710     case LIST:
711     case TIME:
712     case UNKNOWN:
713     default:
714         throw new RuntimeException();
715     }
716 }
717 void gen(int nest, String var) {
718     if (node == null) {
719         for (ConnectionNode n : children)
720             n.gen(nest, var);
721     } else {
722         genNode(nest, var);
723     }
724 }
725
726 private void genNode(int nest, String var) {
727     switch (node.getType().getTypeCode()) {
728     case BOOL:
729         genBool(nest, var);
730         break;
731     case ERROR:
732         genError(nest, var);
733         break;
734     case AUTO:
735         genAuto(nest, var);
736         break;
737     case LIST:
738         genList(nest, var);
739         break;
740     case NUMBER:
```

```
741             genNumber(nest, var);
742             break;
743         case RECORD:
744             genRecord(nest, var);
745             break;
746         case STRING:
747             genString(nest, var);
748             break;
749
750         case TIME:
751         case UNKNOWN:
752         default:
753             throw new RuntimeException();
754         }
755     }
756
757     private void genLeaf(int nest, String var) {
758         for (ConnectionNode link : links) {
759             ++letCount;
760             printf(nest, "let %s = %s in do {\n", link.varName, var);
761         }
762     }
763
764     private void genBool(int nest, String var) {
765         genLeaf(nest, var);
766     }
767
768     private void genError(int nest, String var) {
769         genLeaf(nest, var);
770     }
771
772     private void genAuto(int nest, String var) {
773         for (ConnectionNode n : children)
774             n.gen(nest, var);
775     }
776
777     private void genList(int nest, String var) {
778         String list = getNextConnectionNodeName();
779         if (inControlFlow)
780             printf(nest, "%s <- hoistEither (jsonToList %s);\n", list, var);
781         else
782             printf(nest, "%s <- eitherSIO (jsonToList %s);\n", list, var);
783         for (ConnectionNode link : links) {
784             int realLetCount = letCount;
785             String tmpList = getNextConnectionNodeName();
786             printf(nest, "%s <- flip mapM %s (\\"%s -> do {\n", tmpList, list, varName);
787             for (ConnectionNode n : children) {
788                 n.gen(nest+1, varName);
789             }
790             String res = null;
791             assert link.node.getType().getTypeCode() == TypeCode.LIST;
792             for (ConnectionNode n : link.children) {
793                 res = n.varName;
794                 n.genSubTree(nest+1);
795             }
796         }
797     }
798 }
```

```
796         printf(nest+1, "return %s;\n", res);
797         String brackets = "}";
798         for (int i = realLetCount; i < letCount; i++)
799             brackets += "}";
800         letCount = realLetCount;
801         printf(nest, "%s;\n", brackets);
802         ++letCount;
803         println(nest, "let "+link.varName+" = listToJson "+tmpList+" in do {}");
804     }
805 }
806
807 private void genNumber(int nest, String var) {
808     genLeaf(nest, var);
809 }
810
811 private void genRecord(int nest, String var) {
812     String rec = getNextConnectionNodeName();
813     if (inControlFlow)
814         printf(nest, "%s <- hoistEither (jsonToMap %s);\n", rec, var);
815     else
816         printf(nest, "%s <- eitherSIO (jsonToMap %s);\n", rec, var);
817     for (ConnectionNode c : children) {
818         String tmp = getNextConnectionNodeName();
819         String v = getNextConnectionNodeName();
820         ++letCount;
821         printf(nest, "let %s = Map.lookup \"%s\" %s in do {\n", tmp, c.node.getColumnName
822             (), rec);
823         if (inControlFlow)
824             printf(nest, "%s <- hoistEither (maybeToEither (DoesNotExist \"unable to find
825 key '%s'\") %s);\n", v, c.node.getColumnName(), tmp);
826         else
827             printf(nest, "%s <- eitherSIO (maybeToEither (DoesNotExist \"unable to find
828 key '%s'\") %s);\n", v, c.node.getColumnName(), tmp);
829         c.gen(nest, v);
830     }
831 }
832
833
834 private void printDotReversed(PrintStream out) {
835     if (isPrinted)
836         return;
837     isPrinted = true;
838
839     ConnectionNode p = parent;
840     if (p == null)
841         return;
842     out.println(" "+varName+"_"+getNodeString()+" -> "+p.varName+"_"+p.getNodeString()+"";
843     );
844     p.printDotReversed(out);
845 }
846
847 void resetIsPrinted() {
```

```
847         isPrinted = false;
848         for (ConnectionNode n : children)
849             n.resetIsPrinted();
850     }
851
852     String getNodeString() {
853         if (node == null)
854             return "root";
855         switch (node.getType().getTypeCode()) {
856             case BOOL:
857                 return node.getColumnName() == null ? "bool" : node.getColumnName()+"_bool";
858             case ERROR:
859                 return node.getColumnName() == null ? "error" : node.getColumnName()+"_error";
860             case NUMBER:
861                 return node.getColumnName() == null ? "number" : node.getColumnName()+"_number";
862             case STRING:
863                 return node.getColumnName() == null ? "string" : node.getColumnName()+"_string";
864             case LIST:
865                 return node.getColumnName() == null ? "list" : node.getColumnName()+"_list";
866             case AUTO:
867                 return node.getColumnName() == null ? "auto" : node.getColumnName()+"_auto";
868             case RECORD:
869                 return node.getColumnName() == null ? "record" : node.getColumnName()+"_record";
870
871             case TIME:
872             case UNKNOWN:
873             default:
874                 throw new RuntimeException();
875         }
876     }
877
878     boolean isLeaf() {
879         switch (node.getType().getTypeCode()) {
880             case BOOL:
881             case ERROR:
882             case NUMBER:
883             case STRING:
884                 return true;
885
886             case AUTO:
887             case LIST:
888             case RECORD:
889                 return false;
890
891             case TIME:
892             case UNKNOWN:
893             default:
894                 throw new RuntimeException();
895         }
896     }
897 }
898
899 private class ConnectionTree {
900     ConnectionTree inputTree;
901     ConnectionNode root = new ConnectionNode(null, null);
```

```
902     Map<Integer, ConnectionNode> indexMap = new HashMap<>();
903
904     ConnectionNode insert(List<TypeConnection.PathNode> path, int index) {
905         ConnectionNode ret = insert(root, new LinkedList<>(path));
906         assert ret.isLeaf();
907         indexMap.put(index, ret);
908         return ret;
909     }
910
911     ConnectionNode insert(ConnectionNode parent, LinkedList<TypeConnection.PathNode> path) {
912         if (path.isEmpty()) {
913             return parent;
914         }
915
916         TypeConnection.PathNode pathNode = path.removeFirst();
917         for (ConnectionNode c : parent.children) {
918             if (c.node.equals(pathNode))
919                 return insert(c, path);
920         }
921
922         ConnectionNode n = parent.getNewNodeAfter(pathNode);
923         return insert(n, path);
924     }
925
926     void gen(int nest, String var) {
927         root.gen(nest, var);
928         for (ConnectionNode n : inputTree.root.children)
929             n.genSubTree(nest);
930     }
931
932     @SuppressWarnings("unused")
933     void printDot(String fileName) throws FileNotFoundException {
934         File file = new File(fileName);
935         PrintStream s = new PrintStream(new FileOutputStream(file));
936         s.println("digraph G {");
937         root.printDot(s);
938         s.println("}");
939         root.resetIsPrinted();
940         inputTree.root.resetIsPrinted();
941     }
942
943     void linkWithInputTree(ConnectionTree inputTree, List<TypeConnection.SingleConnection>
conns) {
944         this.inputTree = inputTree;
945         for (TypeConnection.SingleConnection c : conns) {
946             ConnectionNode in = inputTree.indexMap.get(c.getInputIndex());
947             ConnectionNode out = indexMap.get(c.getOutputIndex());
948             out.linkWith(in);
949         }
950         for (ConnectionNode n : root.children)
951             linkListsOutput(n, new Stack<>());
952     }
953
954     private void linkListsOutput(ConnectionNode node, Stack<ConnectionNode> lists) {
955         Stack<ConnectionNode> newList = new Stack<ConnectionNode>();
```

```
956         newLists.addAll(lists);
957         if (node.node.getType().getTypeCode() == TypeCode.LIST)
958             newLists.push(node);
959         if (node.children.size() > 0) {
960             for (ConnectionNode c : node.children) {
961                 linkListsOutput(c, newLists);
962             }
963         } else {
964             for (ConnectionNode link : node.links) {
965                 Stack<ConnectionNode> s = new Stack<>();
966                 s.addAll(newLists);
967                 linkListsInput(link, s);
968             }
969         }
970     }
971
972     private void linkListsInput(ConnectionNode node, Stack<ConnectionNode> lists) {
973         if (node.node == null)
974             return;
975
976         if (node.node.getType().getTypeCode() == TypeCode.LIST)
977             node.linkLabelWith(lists.pop());
978         linkListsInput(node.parent, lists);
979     }
980 }
981
982     private String genTypeConnectionMap(int nest, String var, TypeConnection typeConn) {
983         List<TypeConnection.SingleConnection> conns = typeConn.getConnections();
984         HashMap<Integer, List<TypeConnection.PathNode>> outputPaths = new HashMap<>();
985         HashMap<Integer, List<TypeConnection.PathNode>> inputPaths = new HashMap<>();
986         for (TypeConnection.SingleConnection c : conns) {
987             Integer i = c.getOutputIndex();
988             outputPaths.put(i, typeConn.getIndexPath(i, typeConn.getOutputType()));
989             i = c.getInputIndex();
990             inputPaths.put(i, typeConn.getIndexPath(i, typeConn.getInputType()));
991         }
992         ConnectionTree outputTree = new ConnectionTree();
993         for (Integer k : outputPaths.keySet()) {
994             outputTree.insert(outputPaths.get(k), k);
995         }
996         ConnectionTree inputTree = new ConnectionTree();
997         for (Integer k : inputPaths.keySet()) {
998             inputTree.insert(inputPaths.get(k), k);
999         }
1000         outputTree.linkLabelWithInputTree(inputTree, conns);
1001         outputTree.gen(nest, var);
1002         String ret = null;
1003         for (ConnectionNode n : inputTree.root.children) {
1004             ret = n.varName;
1005         }
1006         return ret;
1007     }
1008
1009     private void genControlArgs(int nest, ControlConnection conn) {
1010         if (isPreprocessing)
```

```
1011         return;
1012     for (ASTCFTypeConnection t : conn.typeConnections) {
1013         String out;
1014         if (t.outputName.equals(ControlPart.errorEvent)) {
1015             out = componentErrorEventMap.get(t.outputComponent);
1016             assert out != null;
1017         } else {
1018             out= genControlVarId(ControlVarIdType.OUTPUT,
1019                                 t.outputComponent.sequenceNumber, t.outputName);
1020         }
1021         String in = genControlVarId(ControlVarIdType.INPUT,
1022                                       t.inputComponent.sequenceNumber, t.inputName);
1023         String tmp = getNextConnectionNodeName();
1024         printf(1, "%s <- lift (runSeqServiceIO (getVarSIO \"%s\")) >>= hoistEither;\n", tmp,
1025                out);
1026         tmp = genTypeConnectionMap(nest, tmp, t.typeConnection);
1027         printf(1, "lift (runSeqServiceIO (setVarSIO \"%s\" %s)) >>= hoistEither;\n", in, tmp);
1028         resetConnectionNodeId();
1029     }
1030 }
1031 @Override
1032 public void visit(ASTCFComponentStop n) {
1033     if (!inControlFlow)
1034         return;
1035     endControlFunc();
1036     Map<String,String> evs = new HashMap<>();
1037     evs.put("Event", "terminate");
1038     genInteraction(currControlFunc, "Runtime.nop", emptyStringStringMap,
1039                   evs, emptyStringStringMap);
1040 }
1041
1042
1043 @Override
1044 public void visit(ASTCFComponentFail n) {
1045     if (!inControlFlow)
1046         return;
1047     endControlFunc();
1048     Map<String,String> args = new HashMap<>();
1049     for (RecordColumn col : n.componentInputs) {
1050         String varName = genControlVarId(ControlVarIdType.INPUT, n.sequenceNumber, col.getId()
1051 );
1052         args.put(col.getId(), varName);
1053     }
1054     Map<String,String> evs = new HashMap<>();
1055     evs.put("Event", "terminate");
1056     genInteraction(currControlFunc, "Runtime.fail", args, evs,
1057                   emptyStringStringMap);
1058 }
1059
1060
1061 @Override
1062 public void visit(ASTCFComponentInteraction n) {
1063     if (!inControlFlow) {
```

```
1064     }
1065     endControlFunc();
1066
1067     Map<String, String> args = new HashMap<>();
1068     for (RecordColumn col : n.componentInputs) {
1069         String varName = genControlVarId(ControlVarIdType.INPUT, n.sequenceNumber, col.getId())
1070     );
1071         args.put(col.getId(), varName);
1072     }
1073     Map<String, String> res = new HashMap<>();
1074     for (RecordColumn col : n.interactionOutputs) {
1075         String varName = genControlVarId(ControlVarIdType.OUTPUT, n.sequenceNumber, col.getId()
1076     );
1077         res.put(col.getId(), varName);
1078     }
1079     Map<String, String> evs = getEventMap(n);
1080     String name = n.interactionPlatform+"."+n.interactionTitle;
1081     genInteraction(currControlFunc, name, args, evs, res);
1082     genContinuation(n);
1083 }
1084
1085 @Override
1086 public void visit(ASTCFComponentStart n) {
1087     if (!inControlFlow)
1088         return;
1089     if (!startControlFunc(n))
1090         return;
1091     startSequenceNumber = n.sequenceNumber;
1092     String evt = null;
1093     for (String e : new HashSet<>(n.componentEventMap.values())) {
1094         evt = e;
1095     }
1096     ControlConnection nextControlConn = controlFlowEventMap.get(new ControlEvent(n, evt));
1097     genControlArgs(1, nextControlConn);
1098     nextControlConn.inputComponent.accept(this);
1099 }
1100
1101 @Override
1102 public void visit(ASTCFEdge n) {
1103     String e = n.startComponent.componentEventMap.get(n.startEventIndex);
1104     controlFlowEventMap.put(
1105         new ControlEvent(n.startComponent, e),
1106         new ControlConnection(n.endComponent, n.typeConnections));
1107 }
1108
1109 @Override
1110 public void visit(ASTDFEdge n) {}
1111
1112 @Override
1113 public void visit(ASTDFComponentInput n) {
1114     if (!inDataFlow)
1115         return;
1116     DataComponentNode dcn = dataComponentTree.nodeMap.get(n);
```

```
1117     String name = getNextConnectionNodeName();
1118     ++letCount;
1119     println(1, "let "+name+" = "+asPatternStr(n.inputTitle)+" in do {");
1120     dcn.outputVars.put(n.componentOutputs.get(0), name);
1121 }
1122
1123 @Override
1124 public void visit(ASTDFComponentOutput n) {
1125     if (!inDataFlow)
1126         return;
1127
1128     DataComponentNode dcn = dataComponentTree.nodeMap.get(n);
1129     for (DataComponentNodeIO pred : dcn.inputEdges.keySet()) {
1130         ASTDFEdge e = dcn.inputEdges.get(pred);
1131         String var = pred.node.outputVars.get(e.outputName);
1132         String res = genTypeConnectionMap(1, var, e.typeConnection);
1133         dataOutputMap.put(n.outputTitle, res);
1134     }
1135 }
1136
1137 @Override
1138 public void visit(ASTDF n) {
1139     String funcId = genFuncId(n);
1140     startDataFunc(funcId, n.dataFlowInputs, n.dataFlowOutputs, n.dataFlowInputOrderFixed);
1141
1142     namePrefix = namePrefix+funcId.substring(funcId.lastIndexOf(nameSepId)));
1143
1144     DataComponentTree tree = new DataComponentTree();
1145     for (ASTDFEdge e : n.dataFlowEdges) {
1146         tree.link(e);
1147     }
1148     resetConnectionNodeId();
1149     dataComponentTree = tree;
1150     tree.prepare();
1151     for (DataComponentNode dcn : tree.roots) {
1152         dcn.start();
1153     }
1154     List<RecordColumn> outputs = sortColumns(n.dataFlowOutputs);
1155     String res = "(";
1156     for (int i = 0; i < outputs.size()-1; i++) {
1157         String name = dataOutputMap.get(outputs.get(i).getId());
1158         res += name+", ";
1159     }
1160     if (outputs.size() > 0) {
1161         String name = dataOutputMap.get(outputs.get(outputs.size()-1).getId());
1162         res += name;
1163     }
1164     res += ")";
1165
1166     endDataFunc(res);
1167
1168     for (ASTDFComponent c : n.dataFlowComponents) {
1169         c.accept(this);
1170     }
1171     namePrefix = namePrefix.substring(0, namePrefix.lastIndexOf(nameSepId));
```

```
1172     }
1173
1174     private Map<String, String> getArgMap(DataComponentNode dcn) {
1175         Map<String, String> argMap = new HashMap<>();
1176         for (DataComponentNodeIO pred : dcn.inputEdges.keySet()) {
1177             ASTDFEdge e = dcn.inputEdges.get(pred);
1178             String var = pred.node.outputVars.get(e.outputName);
1179             String res = genTypeConnectionMap(1, var, e.typeConnection);
1180             argMap.put(e.inputName, res);
1181         }
1182         return argMap;
1183     }
1184
1185     private String getArgString(List<String> inputs, Map<String, String> argMap) {
1186         String arg = "(";
1187         for (int i = 0; i < inputs.size() - 1; i++) {
1188             String name = argMap.get(inputs.get(i));
1189             arg += name + ", ";
1190         }
1191         if (inputs.size() > 0) {
1192             String name = argMap.get(inputs.get(inputs.size() - 1));
1193             arg += name;
1194         }
1195         arg += ")";
1196         return arg;
1197     }
1198
1199     private String getResultStringSetMap(DataComponentNode dcn, ASTDFComponent n) {
1200         for (String out : n.componentOutputs) {
1201             dcn.outputVars.put(out, getNextConnectionNodeName());
1202         }
1203
1204         List<String> outputs = n.componentOutputs;
1205         String res = "(";
1206         for (int i = 0; i < outputs.size() - 1; i++) {
1207             String name = dcn.outputVars.get(outputs.get(i));
1208             res += name + ", ";
1209         }
1210         if (outputs.size() > 0) {
1211             String name = dcn.outputVars.get(outputs.get(outputs.size() - 1));
1212             res += name;
1213         }
1214         res += ")";
1215         return res;
1216     }
1217
1218     public void visit(ASTDFComponentStringOperation n) {
1219         if (!inDataFlow)
1220             return;
1221
1222         DataComponentNode dcn = dataComponentTree.nodeMap.get(n);
1223         Map<String, String> argMap = getArgMap(dcn);
1224         String arg = getArgString(n.componentInputs, argMap);
1225         String res = getResultStringSetMap(dcn, n);
1226     }
```

```
1227     switch (DataPartStringOperation.StringOperationType.valueOf(n.stringOperationType)) {
1228         case CONCATENATE:
1229             println(1, res+" <- dataStringConcatenate "+arg+";");
1230             break;
1231         case CONTAINS:
1232             println(1, res+" <- dataStringContains "+arg+";");
1233             break;
1234         case LENGTH:
1235             println(1, res+" <- dataStringLength "+arg+";");
1236             break;
1237         default:
1238             throw new RuntimeException();
1239     }
1240 }
1241
1242 @Override
1243 public void visit(ASTDFComponentArithmetic n) {
1244     if (!inDataFlow)
1245         return;
1246
1247     DataComponentNode dcn = dataComponentTree.nodeMap.get(n);
1248     Map<String, String> argMap = getArgMap(dcn);
1249     String arg = getArgString(n.componentInputs, argMap);
1250     String res = getResultStringSetMap(dcn, n);
1251
1252     switch (DataPartArithmetic.ArithmeticType.valueOf(n.arithmeticType)) {
1253         case DIV:
1254             println(1, res+" <- dataDiv "+arg+";");
1255             break;
1256         case MINUS:
1257             println(1, res+" <- dataMinus "+arg+";");
1258             break;
1259         case MUL:
1260             println(1, res+" <- dataMul "+arg+";");
1261             break;
1262         case NEGATE:
1263             println(1, res+" <- dataNegate "+arg+";");
1264             break;
1265         case PLUS:
1266             println(1, res+" <- dataPlus "+arg+";");
1267             break;
1268         case AND:
1269             println(1, res+" <- dataAnd "+arg+";");
1270             break;
1271         case OR:
1272             println(1, res+" <- dataOr "+arg+";");
1273             break;
1274         case NOT:
1275             println(1, res+" <- dataNot "+arg+";");
1276             break;
1277         default:
1278             throw new RuntimeException();
1279     }
1280 }
1281 }
```

```
1282     private String genCastFromNumber(int nest, Type to, String var) {
1283         while (to.getTypeCode() == TypeCode.AUTO)
1284             to = ((TypeAuto)to).getType();
1285         switch (to.getTypeCode()) {
1286             case STRING:
1287                 String ret = getNextConnectionNodeName();
1288                 ++letCount;
1289                 println(nest, "let "+ret+" = strToJson (drop 7 (show "+var+")) in do {");
1290                 return ret;
1291             case NUMBER:
1292                 return var;
1293             case BOOL:
1294             case ERROR:
1295             case LIST:
1296             case RECORD:
1297             case TIME:
1298             case UNKNOWN:
1299             case AUTO:
1300             default:
1301                 throw new RuntimeException();
1302         }
1303     }
1304 }
1305
1306 private String genCastFromString(int nest, Type to, String var) {
1307     while (to.getTypeCode() == TypeCode.AUTO)
1308         to = ((TypeAuto)to).getType();
1309     switch (to.getTypeCode()) {
1310         case STRING:
1311             return var;
1312         case NUMBER:
1313             String retn = getNextConnectionNodeName();
1314             String tmp = getNextConnectionNodeName();
1315             println(nest, tmp+"  
- eitherSIO (jsonToStr "+var+");");
1316             println(nest, retn+"  
- eitherSIO $ case readMaybe "+tmp+" :: Maybe Scientific of {");
1317             println(nest+1, "Just x -> Right (scientificToJson x);");
1318             println(nest+1, "_ -> Left (InvalidArgument $ \"Failed cast from String to Number\");");
1319         );
1320             println(nest, "});");
1321             return retn;
1322         case BOOL:
1323             String retb = getNextConnectionNodeName();
1324             println(nest, retb+"  
- eitherSIO $ case "+var+" of {");
1325             println(nest+1, "\"True\" -> Right (boolToJson True);");
1326             println(nest+1, "\"False\" -> Right (boolToJson False);");
1327             println(nest+1, "_ -> Left (InvalidArgument $ \"Failed cast from String to Boolean\");");
1328         );
1329             println(nest, "});");
1330             return retb;
1331         case ERROR:
1332         case LIST:
1333         case RECORD:
1334         case TIME:
1335         case UNKNOWN:
```

```
1335     case AUTO:
1336     default:
1337         throw new RuntimeException();
1338     }
1339 }
1340
1341 private String genCastFromBool(int nest, Type to, String var) {
1342     while (to.getTypeCode() == TypeCode.AUTO)
1343         to = ((TypeAuto)to).getType();
1344     switch (to.getTypeCode()) {
1345     case STRING:
1346         String ret = getNextConnectionNodeName();
1347         ++letCount;
1348         println(nest, "let "+ret+" = strToJson (drop 5 (show "+var+")) in do {}");
1349         return ret;
1350     case BOOL:
1351         return var;
1352     case NUMBER:
1353     case ERROR:
1354     case LIST:
1355     case RECORD:
1356     case TIME:
1357     case UNKNOWN:
1358     case AUTO:
1359     default:
1360         throw new RuntimeException();
1361     }
1362 }
1363
1364
1365 private String genCastFromRecord(int nest, TypeRecord from, Type to, String var) {
1366     while (to.getTypeCode() == TypeCode.AUTO)
1367         to = ((TypeAuto)to).getType();
1368     TypeRecord toRec = (TypeRecord)to;
1369
1370     String fromMap = getNextConnectionNodeName();
1371     String toMap = getNextConnectionNodeName();
1372     printf(nest, "%s <- eitherSIO (jsonToMap %s);\n", fromMap, var);
1373     ++letCount;
1374     println(nest, "let "+toMap+" = Map.empty :: Map.HashMap Text.Text Json in do {}");
1375     List<RecordColumn> fcols = from.getColumnsInOrder();
1376     List<RecordColumn> tcols = toRec.getColumnsInOrder();
1377
1378     for (int i = 0; i < fcols.size(); i++) {
1379         RecordColumn fcol = fcols.get(i);
1380         String f = getNextConnectionNodeName();
1381         printf(nest, "%s <- eitherSIO (jsonObjLookup \"%s\" %s);\n", f, fcol.getId(), fromMap)
1382         ;
1383         String t = genCast(nest, fcol.getType(), tcols.get(i).getType(), f);
1384         String tmpToMap = getNextConnectionNodeName();
1385         ++letCount;
1386         printf(nest, "let %s = Map.insert \"%s\" %s %s in do {\n",
1387             tmpToMap, fcol.getId(), t, toMap);
1388         toMap = tmpToMap;
1389     }
1390 }
```

```
1389
1390     String ret = getNextConnectionNodeName();
1391     ++letCount;
1392     println(nest, "let "+ret+" = mapToJson "+toMap+" in do {");
1393     return ret;
1394 }
1395
1396 private String genCastFromList(int nest, TypeList from, Type to, String var) {
1397     while (to.getTypeCode() == TypeCode.AUTO)
1398         to = ((TypeAuto)to).getType();
1399     TypeList toList = (TypeList)to;
1400
1401     String list = getNextConnectionNodeName();
1402     String tmp = getNextConnectionNodeName();
1403     String next = getNextConnectionNodeName();
1404     int realLetCount = letCount;
1405     println(nest, list+"< - eitherSIO (jsonToList "+var+");");
1406     println(nest, tmp+"< - flip mapM "+list+" (\\""+next+" -> do {");
1407     String sub = genCast(nest+1, from.getChildType(), toList.getChildType(), next);
1408     println(nest+1, "return "+sub+";");
1409     String brackets = "}";
1410     for (int i = realLetCount; i < letCount; i++)
1411         brackets += "}";
1412     letCount = realLetCount;
1413     println(nest, brackets+");");
1414     String ret = getNextConnectionNodeName();
1415     ++letCount;
1416     println(nest, "let "+ret+" = listToJson "+tmp+" in do {");
1417     return ret;
1418 }
1419
1420 private String genCast(int nest, Type from, Type to, String var) {
1421     while (from.getTypeCode() == TypeCode.AUTO)
1422         from = ((TypeAuto)from).getType();
1423     switch (from.getTypeCode()) {
1424     case BOOL:
1425         return genCastFromBool(nest, to, var);
1426     case LIST:
1427         return genCastFromList(nest, (TypeList)from, to, var);
1428     case NUMBER:
1429         return genCastFromNumber(nest, to, var);
1430     case RECORD:
1431         return genCastFromRecord(nest, (TypeRecord)from, to, var);
1432     case STRING:
1433         return genCastFromString(nest, to, var);
1434
1435     case ERROR:
1436     case TIME:
1437     case UNKNOWN:
1438     default:
1439         throw new RuntimeException();
1440     }
1441 }
1442
1443 @Override
```

```
1444     public void visit(ASTDFComponentCast n) {
1445         if (!inDataFlow)
1446             return;
1447
1448         DataComponentNode dcn = dataComponentTree.nodeMap.get(n);
1449         for (DataComponentNodeIO pred : dcn.inputEdges.keySet()) {
1450             ASTDFEdge e = dcn.inputEdges.get(pred);
1451             String var = pred.node.outputVars.get(e.outputName);
1452             String tmp = genTypeConnectionMap(1, var, e.typeConnection);
1453             dcn.outputVars.put(n.componentOutputs.get(0),
1454                 genCast(1, n.castFrom, n.castTo, tmp));
1455         }
1456     }
1457
1458     @Override
1459     public void visit(ASTDFComponentComparison n) {
1460         if (!inDataFlow)
1461             return;
1462
1463         DataComponentNode dcn = dataComponentTree.nodeMap.get(n);
1464         Map<String, String> argMap = getArgMap(dcn);
1465         String arg = getArgString(n.componentInputs, argMap);
1466         String res = getResultStringSetMap(dcn, n);
1467
1468         switch (DataPartComparison.ComparisonType.valueOf(n.comparisonType)) {
1469             case EQUAL:
1470                 println(1, res+" <- dataEqual "+arg+";");
1471                 break;
1472             case NOTEQUAL:
1473                 println(1, res+" <- dataNotEqual "+arg+");");
1474                 break;
1475             case GREATER:
1476                 println(1, res+" <- dataGreater "+arg+");");
1477                 break;
1478             case GREATEREQUAL:
1479                 println(1, res+" <- dataGreaterEqual "+arg+");");
1480                 break;
1481             case LESS:
1482                 println(1, res+" <- dataLess "+arg+");");
1483                 break;
1484             case LESSEQUAL:
1485                 println(1, res+" <- dataLessEqual "+arg+");");
1486                 break;
1487             default:
1488                 throw new RuntimeException();
1489         }
1490     }
1491
1492     private String genBoolConstant(ConstantBool c) {
1493         String ret = getNextConnectionNodeName();
1494         ++letCount;
1495         printf(1, "let %s = boolToJson %s in do {\n", ret, c.getValue() ? "True" : "False");
1496         return ret;
1497     }
1498 }
```

```
1499     private String genNumberConstant(ConstantNumber c) {
1500         String ret = getNextConnectionNodeName();
1501         ++letCount;
1502         println(1, "let "+ret+" = scientificToJson (read \\""+c.getValue()+"\") in do {");
1503         return ret;
1504     }
1505
1506     private String genStringConstant(ConstantString c) {
1507         String ret = getNextConnectionNodeName();
1508         String str = StringEscapeUtils.escapeJava(c.getValue());
1509         ++letCount;
1510         println(1, "let "+ret+" = strToJson \\""+str+"\\" in do {");
1511         return ret;
1512     }
1513
1514     private String genListConstant(ConstantList c) {
1515         ArrayList<String> vars = new ArrayList<>();
1516         for (Constant child : c.getValues()) {
1517             vars.add(genConstant(child));
1518         }
1519         String ret = getNextConnectionNodeName();
1520         String val = "[";
1521         for (int i = 0; i < vars.size()-1; i++) {
1522             val += vars.get(i) + ", ";
1523         }
1524         if (vars.size() > 0) {
1525             val += vars.get(vars.size()-1);
1526         }
1527         val += "]";
1528         ++letCount;
1529         println(1, "let "+ret+" = listToJson "+val+" in do {");
1530         return ret;
1531     }
1532
1533     private String genRecordConstant(ConstantRecord c) {
1534         String[] rets = new String[2];
1535         rets[0] = getNextConnectionNodeName();
1536         rets[1] = getNextConnectionNodeName();
1537         ++letCount;
1538         println(1, "let "+rets[0]+" = Map.empty :: Map.HashMap Text.Text Json in do {");
1539         List<RecordValue> cols = c.getColumnsInOrder();
1540         for (int i = 0; i < cols.size(); i++) {
1541             String val = genConstant(cols.get(i).getConstant());
1542             ++letCount;
1543             println(1, "let "+rets[(i+1)%2]+" = Map.insert \\""+cols.get(i).getId()+"\" " +
1544                     val+" "+rets[i%2]+" in do {");
1545         }
1546         String ret = getNextConnectionNodeName();
1547         ++letCount;
1548         println(1, "let "+ret+" = mapToJson "+rets[cols.size()%2]+" in do {");
1549         return ret;
1550     }
1551
1552     private String genConstant(Constant c) {
1553         switch (c.getType().getTypeCode()) {
```

```
1554     case BOOL:
1555         return genBoolConstant((ConstantBool)c);
1556     case LIST:
1557         return genListConstant((ConstantList)c);
1558     case NUMBER:
1559         return genNumberConstant((ConstantNumber)c);
1560     case RECORD:
1561         return genRecordConstant((ConstantRecord)c);
1562     case STRING:
1563         return genStringConstant((ConstantString)c);
1564
1565     case ERROR:
1566     case AUTO:
1567     case TIME:
1568     case UNKNOWN:
1569     default:
1570         throw new RuntimeException();
1571     }
1572 }
1573
1574 @Override
1575 public void visit(ASTDFComponentRecordConstructor n) {
1576     if (!inDataFlow)
1577         return;
1578
1579     DataComponentNode dcn = dataComponentTree.nodeMap.get(n);
1580
1581     String map = getNextConnectionNodeName();
1582     ++letCount;
1583     println(1, "let "+map+" = Map.empty :: Map.HashMap Text.Text Json in do {");
1584
1585     for (DataComponentNodeIO pred : dcn.inputEdges.keySet()) {
1586         ASTDFEdge e = dcn.inputEdges.get(pred);
1587         String var = pred.node.outputVars.get(e.outputName);
1588         String tmp = genTypeConnectionMap(1, var, e.typeConnection);
1589         String tmpMap = getNextConnectionNodeName();
1590         ++letCount;
1591         println(1, "let "+tmpMap+" = Map.insert '"+e.inputName+"' '"+tmp+"' "+map+" in do {");
1592         map = tmpMap;
1593     }
1594
1595     String result = getNextConnectionNodeName();
1596     ++letCount;
1597     println(1, "let "+result+" = mapToJson "+map+" in do {");
1598     dcn.outputVars.put(n.componentOutputs.get(0), result);
1599 }
1600
1601 @Override
1602 public void visit(ASTDFComponentConstant n) {
1603     if (!inDataFlow)
1604         return;
1605
1606     DataComponentNode dcn = dataComponentTree.nodeMap.get(n);
1607     dcn.outputVars.put(n.componentOutputs.get(0), genConstant(n.constantValue));
1608 }
```

```
1609
1610     @Override
1611     public void visit(ASTDFComponentStringFormat n) {
1612         if (!inDataFlow)
1613             return;
1614
1615         DataComponentNode dcn = dataComponentTree.nodeMap.get(n);
1616         Map<String, String> argMap = new HashMap<>();
1617         for (DataComponentNodeIO pred : dcn.inputEdges.keySet()) {
1618             ASTDFEdge e = dcn.inputEdges.get(pred);
1619             String var = pred.node.outputVars.get(e.outputName);
1620             String tmp = genTypeConnectionMap(1, var, e.typeConnection);
1621             argMap.put(e.inputName, tmp);
1622         }
1623
1624         String args = "[";
1625         for (String in : n.componentInputs) {
1626             String tmp = argMap.get(in);
1627             args += tmp+", ";
1628         }
1629         if (n.componentInputs.size() > 0)
1630             args = args.substring(0, args.length()-2);
1631         args += "]";
1632
1633         String fmt = StringEscapeUtils.escapeJava(n.stringFormat);
1634         String res = getNextConnectionNodeName();
1635         println(1, res+" <- dataStringFormat \"\""+fmt+"\\" " +args+";");
1636
1637         dcn.outputVars.put(n.componentOutputs.get(0), res);
1638     }
1639
1640
1641     @Override
1642     public void visit(ASTDFComponentListOperation n) {
1643         if (!inDataFlow)
1644             return;
1645
1646         DataComponentNode dcn = dataComponentTree.nodeMap.get(n);
1647         Map<String, String> argMap = getArgMap(dcn);
1648         String arg = getArgString(n.componentInputs, argMap);
1649         String res = getResultStringSetMap(dcn, n);
1650
1651         switch (DataPartListOperation.ListOperationType.valueOf(n.listOperationType)) {
1652             case LENGTH:
1653                 println(1, res+" <- dataLength "+arg+";");
1654                 break;
1655             case APPEND:
1656                 println(1, res+" <- dataAppend "+arg+";");
1657                 break;
1658             case PREPEND:
1659                 println(1, res+" <- dataPrepend "+arg+";");
1660                 break;
1661             case ALLTRUE:
1662                 println(1, res+" <- dataAllTrue "+arg+" actID;");
1663                 break;

```

```
1664     case ANYTRUE:
1665         println(1, res+" <- dataAnyTrue "+arg+" actID;");
1666         break;
1667     case SUM:
1668         println(1, res+" <- dataSum "+arg+" actID;");
1669         break;
1670     case CONTAINS:
1671         println(1, res+" <- dataContains "+arg+" actID;");
1672         break;
1673     case AUGMENT:
1674         println(1, res+" <- dataAugment "+arg+" actID;");
1675         break;
1676     default:
1677         throw new RuntimeException();
1678     }
1679 }
1680
1681 @Override
1682 public void visit(ASTDFComponentDF n) {
1683     if (!inDataFlow) {
1684         n.dataFlow.accept(this);
1685         return;
1686     }
1687
1688     DataComponentNode dcn = dataComponentTree.nodeMap.get(n);
1689     List<RecordColumn> inputs = sortColumns(n.dataFlow.dataFlowInputs);
1690     Map<String, String> inputNames = new HashMap<>();
1691     for (DataComponentNodeIO pred : dcn.inputEdges.keySet()) {
1692         ASTDFEdge e = dcn.inputEdges.get(pred);
1693         String var = pred.node.outputVars.get(e.outputName);
1694         String res = genTypeConnectionMap(1, var, e.typeConnection);
1695         inputNames.put(e.inputName, res);
1696     }
1697     String args = "(";
1698     for (int i = 0; i < inputs.size()-1; i++) {
1699         String name = inputs.get(i).getId();
1700         args += inputNames.get(name) + ", ";
1701     }
1702     if (inputs.size() > 0) {
1703         String name = inputs.get(inputs.size()-1).getId();
1704         args += inputNames.get(name);
1705     }
1706     args += ")";
1707
1708     List<RecordColumn> outputs = sortColumns(n.dataFlow.dataFlowOutputs);
1709     for (RecordColumn col : outputs) {
1710         String res = getNextConnectionNodeName();
1711         dcn.outputVars.put(col.getId(), res);
1712     }
1713     String res = "(";
1714     for (int i = 0; i < outputs.size()-1; i++) {
1715         String name = outputs.get(i).getId();
1716         res += dcn.outputVars.get(name) + ", ";
1717     }
1718     if (outputs.size() > 0) {
```

```
1719         String name = outputs.get(outputs.size()-1).getId();
1720         res += dcn.outputVars.get(name);
1721     }
1722     res += ")";
1723     println(1, res+" <- "+genFuncId(n)+" "+args+" actID;");
1724 }
1725
1726 private void genFunctionalFilter(ASTDFComponentFunctional n) {
1727     DataComponentNode dcn = dataComponentTree.nodeMap.get(n);
1728     for (DataComponentNodeIO pred : dcn.inputEdges.keySet()) {
1729         ASTDFEdge e = dcn.inputEdges.get(pred);
1730         String var = pred.node.outputVars.get(e.outputName);
1731         String mapVar = genTypeConnectionMap(1, var, e.typeConnection);
1732         String res = getNextConnectionNodeName();
1733         printf(1, "%s <- dataFilter %s %s actID;\n",
1734                res, genFuncId(n.dataFlow), mapVar);
1735         dcn.outputVars.put(e.inputName, res);
1736     }
1737 }
1738
1739 private void genFunctionalMap(ASTDFComponentFunctional n) {
1740     DataComponentNode dcn = dataComponentTree.nodeMap.get(n);
1741     for (DataComponentNodeIO pred : dcn.inputEdges.keySet()) {
1742         ASTDFEdge e = dcn.inputEdges.get(pred);
1743         String var = pred.node.outputVars.get(e.outputName);
1744         String mapVar = genTypeConnectionMap(1, var, e.typeConnection);
1745         String res = getNextConnectionNodeName();
1746         printf(1, "%s <- dataMap %s %s actID;\n",
1747                res, genFuncId(n.dataFlow), mapVar);
1748         dcn.outputVars.put(e.inputName, res);
1749     }
1750 }
1751
1752 private void genFunctionalJoin(ASTDFComponentFunctional n) {
1753     DataComponentNode dcn = dataComponentTree.nodeMap.get(n);
1754     HashMap<String, String> inputMap = new HashMap<>();
1755     for (DataComponentNodeIO pred : dcn.inputEdges.keySet()) {
1756         ASTDFEdge e = dcn.inputEdges.get(pred);
1757         String var = pred.node.outputVars.get(e.outputName);
1758         String mapVar = genTypeConnectionMap(1, var, e.typeConnection);
1759         inputMap.put(e.inputName, mapVar);
1760     }
1761     List<String> inputs = n.componentInputs;
1762     String args = "(";
1763     for (int i = 0; i < inputs.size()-1; i++) {
1764         String name = inputs.get(i);
1765         args += inputMap.get(name) + ", ";
1766     }
1767     if (inputs.size() > 0) {
1768         String name = inputs.get(inputs.size()-1);
1769         args += inputMap.get(name);
1770     }
1771     args += ")";
1772     String res = getNextConnectionNodeName();
1773     println(1, res+" <- dataJoin "+genFuncId(n.dataFlow)+" "+args+" actID;");
```

```
1774         dcn.outputVars.put(n.componentOutputs.get(0), res);
1775     }
1776
1777     private void genFunctionalForEach(ASTDFComponentFunctional n) {
1778         DataComponentNode dcn = dataComponentTree.nodeMap.get(n);
1779         for (DataComponentNodeIO pred : dcn.inputEdges.keySet()) {
1780             ASTDFEdge e = dcn.inputEdges.get(pred);
1781             String var = pred.node.outputVars.get(e.outputName);
1782             String mapVar = genTypeConnectionMap(1, var, e.typeConnection);
1783             printf(1, "dataForEach %s %s actID;\n",
1784                   genFuncId(n.dataFlow), mapVar);
1785         }
1786     }
1787
1788     private void genFunctionalAccumulate(ASTDFComponentFunctional n) {
1789         DataComponentNode dcn = dataComponentTree.nodeMap.get(n);
1790         HashMap<String, String> inputMap = new HashMap<>();
1791         for (DataComponentNodeIO pred : dcn.inputEdges.keySet()) {
1792             ASTDFEdge e = dcn.inputEdges.get(pred);
1793             String var = pred.node.outputVars.get(e.outputName);
1794             String mapVar = genTypeConnectionMap(1, var, e.typeConnection);
1795             inputMap.put(e.inputName, mapVar);
1796         }
1797         List<String> inputs = n.componentInputs;
1798         String args = "(";
1799         for (int i = 0; i < inputs.size()-1; i++) {
1800             String name = inputs.get(i);
1801             args += inputMap.get(name) + ", ";
1802         }
1803         if (inputs.size() > 0) {
1804             String name = inputs.get(inputs.size()-1);
1805             args += inputMap.get(name);
1806         }
1807         args += ")";
1808         String res = getNextConnectionNodeName();
1809         println(1, res+" <- dataAccumulate "+genFuncId(n.dataFlow)+" "+args+" actID;");
1810         dcn.outputVars.put(n.componentOutputs.get(0), res);
1811     }
1812
1813     @Override
1814     public void visit(ASTDFComponentFunctional n) {
1815         if (!inDataFlow) {
1816             n.dataFlow.accept(this);
1817             return;
1818         }
1819
1820         switch (DataPartFunctional.FunctionalType.valueOf(n.functionalType)) {
1821             case FILTER:
1822                 genFunctionalFilter(n);
1823                 break;
1824             case MAP:
1825                 genFunctionalMap(n);
1826                 break;
1827             case JOIN:
1828                 genFunctionalJoin(n);
```

```
1829         break;
1830     case FOREACH:
1831         genFunctionalForEach(n);
1832         break;
1833     case ACCUMULATE:
1834         genFunctionalAccumulate(n);
1835         break;
1836     default:
1837         throw new RuntimeException();
1838     }
1839 }
1840
1841 @Override
1842 public void visit(ASTDFComponentSink n) {}
1843 }
```

LISTING E.51: lang/ast/ASTVisitorGen.java

```
1 package lang.type;
2
3 public class TypeList implements Type {
4     private Type type;
5
6     public TypeList(Type t) {
7         type = t;
8     }
9
10    public Type getChildType() {
11        return type;
12    }
13
14    @Override
15    public boolean hasAuto() {
16        return type.hasAuto();
17    }
18
19    @Override
20    public boolean isConcrete(boolean b) {
21        return type.isConcrete(b);
22    }
23
24    @Override
25    public boolean equals(Object oth) {
26        if (!(oth instanceof TypeList))
27            return defaultEquals(oth);
28        return ((TypeList)oth).type.equals(type);
29    }
30
31    @Override
32    public int hashCode() {
33        int c = type.hashCode();
34        return c ^ TypeCode.LIST.hashCode();
35    }
36
37    @Override
```

```
38     public String toString() {
39         return "List("+type+ ")";
40     }
41
42     @Override
43     public String getShortDescription() {
44         return "List(...)";
45     }
46
47     @Override
48     public TypeCode getTypeCode() {
49         return TypeCode.LIST;
50     }
51 }
```

LISTING E.52: lang/type/TypeList.java

```
1 package lang.type;
2
3 public interface TypeAutoOutputListener {
4     void autoTypeOutputRemoved();
5 }
```

LISTING E.53: lang/type/TypeAutoOutputListener.java

```
1 package lang.type;
2
3 public class TypeAuto implements Type {
4     private Type type;
5
6     public Type getType() {
7         return type;
8     }
9
10    public TypeAuto() {
11        type = new TypeUnknown("Undefined auto type");
12    }
13
14    public TypeAuto(Type type) {
15        this.type = type;
16    }
17
18    @Override
19    public boolean hasAuto() {
20        return true;
21    }
22
23    @Override
24    public boolean isConcrete(boolean autoIsConcrete) {
25        return autoIsConcrete || type.isConcrete(autoIsConcrete);
26    }
27
28    @Override
29    public TypeCode getTypeCode() {
```

```

30         return TypeCode.AUTO;
31     }
32
33     @Override
34     public String getShortDescription() {
35         return "Auto";
36     }
37
38     @Override
39     public int hashCode() {
40         return type.hashCode();
41     }
42
43     @Override
44     public boolean equals(Object obj) {
45         return type.equals(obj);
46     }
47
48     @Override
49     public String toString() {
50         return "Auto("+type+ ")";
51     }
52 }
```

LISTING E.54: lang/type/TypeAuto.java

```

1 package lang.type;
2
3 public enum TypeCode {UNKNOWN, BOOL, NUMBER, STRING, TIME, RECORD, LIST, ERROR, AUTO};
```

LISTING E.55: lang/type/TypeCode.java

```

1 package lang.type;
2
3 @SuppressWarnings("serial")
4 public class TypeAutoChangeException extends Exception {
5     public TypeAutoChangeException(String msg) {
6         super(msg);
7     }
8 }
```

LISTING E.56: lang/type/TypeAutoChangeException.java

```

1 package lang.type;
2
3 public class TypeTime implements Type {
4     @Override
5     public boolean hasAuto() {
6         return false;
7     }
8
9     @Override
10    public boolean isConcrete(boolean b) {
```

```
11         return true;
12     }
13
14     @Override
15     public boolean equals(Object oth) {
16         if (oth instanceof TypeTime)
17             return true;
18         return defaultEquals(oth);
19     }
20
21     @Override
22     public int hashCode() {
23         return TypeCode.TIME.hashCode();
24     }
25
26     @Override
27     public String toString() {
28         return "Time";
29     }
30
31     @Override
32     public TypeCode getTypeCode() {
33         return TypeCode.TIME;
34     }
35
36     @Override
37     public String getShortDescription() {
38         return toString();
39     }
40 }
```

LISTING E.57: lang/type/TypeTime.java

```
1 package lang.type;
2
3 public class TypeUnknown implements Type {
4     private String message;
5
6     public TypeUnknown(String shortMessage) {
7         message = shortMessage;
8     }
9
10    public String getMessage() {
11        return message;
12    }
13
14    @Override
15    public boolean hasAuto() {
16        return false;
17    }
18
19    @Override
20    public boolean isConcrete(boolean b) {
21        return false;
22    }
23 }
```

```

23
24     @Override
25     public boolean equals(Object oth) {
26         if (oth instanceof TypeUnknown)
27             return ((TypeUnknown) oth).message.equals(message);
28         return defaultEquals(oth);
29     }
30
31     @Override
32     public int hashCode() {
33         return message.hashCode() ^ TypeCode.UNKNOWN.hashCode();
34     }
35
36     @Override
37     public String toString() {
38         //return "Unknown(" + message + ")";
39         return getShortDescription();
40     }
41
42     @Override
43     public TypeCode getTypeCode() {
44         return TypeCode.UNKNOWN;
45     }
46
47     @Override
48     public String getShortDescription() {
49         return "Unknown";
50     }
51 }
```

LISTING E.58: lang/type/TypeUnknown.java

```

1 package lang.type;
2
3 import java.util.ArrayList;
4 import java.util.HashMap;
5 import java.util.List;
6 import java.util.Map;
7 import java.util.Set;
8
9 public class TypeRecord implements Type {
10     private Map<String, Type> columns;
11     private List<RecordColumn> columnsInOrder;
12
13     public TypeRecord(List<RecordColumn> columns) {
14         columnsInOrder = new ArrayList<>(columns);
15
16         this.columns = new HashMap<>();
17         for (RecordColumn c : columns) {
18             if (c.getId().equals(""))
19                 throw new IllegalArgumentException("missing column name");
20             Type prev = this.columns.put(c.getId(), c.getType());
21             if (prev != null)
22                 throw new IllegalArgumentException("duplicate column name " + c.getId() + "'");
23         }
24     }
25 }
```

```
24     }
25
26     public List<RecordColumn> getColumnsInOrder() {
27         return new ArrayList<>(columnsInOrder);
28     }
29
30     public Map<String,Type> getColumns() {
31         return new HashMap<>(this.columns);
32     }
33
34     @Override
35     public boolean hasAuto() {
36         return columns.values().stream().anyMatch((t) -> t.hasAuto());
37     }
38
39     @Override
40     public boolean isConcrete(boolean b) {
41         return columns.values().stream().allMatch((t) -> t.isConcrete(b));
42     }
43
44     @Override
45     public boolean equals(Object obj) {
46         if (!(obj instanceof TypeRecord))
47             return defaultEquals(obj);
48
49         TypeRecord other = (TypeRecord)obj;
50         if (other.columns.size() != columns.size())
51             return false;
52
53         for (String key : columns.keySet()) {
54             Type othType = other.columns.get(key);
55             if (othType == null)
56                 return false;
57             Type type = columns.get(key);
58             if (!othType.equals(type))
59                 return false;
60         }
61         return true;
62     }
63
64     @Override
65     public int hashCode() {
66         return
67             columns.keySet().stream()
68                 .map( Object::hashCode)
69                 .reduce((x,y) -> x^y).get() ^
70             columns.values().stream()
71                 .map( Object::hashCode)
72                 .reduce((x,y) -> x^y).get() ^
73             TypeCode.RECORD.hashCode();
74     }
75
76     @Override
77     public String toString() {
78         StringBuilder b = new StringBuilder();
```

```
79     b.append("Row(");
80     Set<String> keys = columns.keySet();
81     int len = keys.size();
82     int i = 1;
83     for (String key : keys) {
84         b.append(key);
85         b.append(":");
86         b.append(columns.get(key).toString());
87         if (i < len)
88             b.append(", ");
89         i += 1;
90     }
91     b.append(")");
92     return b.toString();
93 }
94
95 @Override
96 public TypeCode getTypeCode() {
97     return TypeCode.RECORD;
98 }
99
100 @Override
101 public String getShortDescription() {
102     return "Record(...)";
103 }
104 }
```

LISTING E.59: lang/type/TypeRecord.java

```
1 package lang.type;
2
3 public class TypeError implements Type {
4     @Override
5     public boolean hasAuto() {
6         return false;
7     }
8
9     @Override
10    public boolean isConcrete(boolean b) {
11        return true;
12    }
13
14    @Override
15    public TypeCode getTypeCode() {
16        return TypeCode.ERROR;
17    }
18
19    @Override
20    public String getShortDescription() {
21        return "Error";
22    }
23
24    @Override
25    public int hashCode() {
26        return getTypeCode().hashCode();
```

```
27     }
28
29     @Override
30     public boolean equals(Object obj) {
31         if (obj instanceof TypeError)
32             return true;
33         return defaultEquals(obj);
34     }
35
36     @Override
37     public String toString() {
38         return getShortDescription();
39     }
40 }
```

LISTING E.60: lang/type/TypeError.java

```
1 package lang.type;
2
3 public interface TypeChangedListener {
4     public void typeChanged(Type newType);
5 }
```

LISTING E.61: lang/type/TypeChangedListener.java

```
1 package lang.type;
2
3 public class RecordColumn {
4     private String id;
5     private Type type;
6     public RecordColumn(String id, Type type) {
7         this.id = id;
8         this.type = type;
9     }
10
11    public String getId() {
12        return id;
13    }
14
15    public Type getType() {
16        return type;
17    }
18 }
```

LISTING E.62: lang/type/RecordColumn.java

```
1 package lang.type;
2
3 public interface Type {
4     boolean isConcrete(boolean autoIsConcrete);
5     boolean hasAuto();
6     TypeCode getTypeCode();
7     String getShortDescription();
```

```

8
9     default boolean defaultEquals(Object obj) {
10        if (obj instanceof TypeAuto)
11            return equals(((TypeAuto)obj).getType());
12        return false;
13    }
14 }
```

LISTING E.63: lang/type/Type.java

```

1 package lang.type;
2
3 import java.util.Collections;
4 import java.util.HashMap;
5 import java.util.LinkedList;
6 import java.util.List;
7 import java.util.ArrayList;
8 import java.util.Map;
9
10 public class TypeConnection {
11     private Type inputType;
12     private Type outputType;
13     private ArrayList<SingleConnection> connections = new ArrayList<>();
14     private TypeAutoInputListener autoListener;
15     private boolean autoConnect;
16
17     public static class PathNode {
18         private String columnName;
19         private Type type;
20         public PathNode(String columnName, Type type) {
21             this.columnName = columnName;
22             this.type = type;
23         }
24
25         public void setColumnName(String col) {
26             this.columnName = col;
27         }
28
29         public String getColumnName() {
30             return columnName;
31         }
32
33         public Type getType() {
34             return type;
35         }
36
37         @Override
38         public boolean equals(Object obj) {
39             if (!(obj instanceof PathNode))
40                 return false;
41             PathNode oth = (PathNode)obj;
42             if (columnName == null || oth.columnName == null)
43                 return columnName == oth.columnName && type.equals(oth.type);
44             return columnName.equals(oth.columnName) && type.equals(oth.type);
45         }
}
```

```
46
47     @Override
48     public int hashCode() {
49         if (columnName == null)
50             return type.hashCode();
51         return columnName.hashCode() ^ type.hashCode();
52     }
53 }
54
55     private static class Connection {
56         int outputIndex;
57         int inputIndex;
58         public Connection(int outIndex, int inIndex) {
59             outputIndex = outIndex;
60             inputIndex = inIndex;
61         }
62
63         public int getOutputIndex() {
64             return outputIndex;
65         }
66
67         public int getInputIndex() {
68             return inputIndex;
69         }
70
71     @Override
72     public boolean equals(Object o) {
73         if (o instanceof Connection) {
74             Connection other = (Connection)o;
75             return other.outputIndex == outputIndex &&
76                 other.inputIndex == inputIndex;
77         }
78         return false;
79     }
80
81     @Override
82     public int hashCode() {
83         return new Integer(outputIndex).hashCode() ^
84             new Integer(inputIndex).hashCode();
85     }
86
87     @Override
88     public String toString() {
89         return "("+outputIndex+" -> "+inputIndex+ ")";
90     }
91 }
92
93     public static class SingleConnection extends Connection {
94         public SingleConnection(int outIndex, int inIndex) {
95             super(outIndex, inIndex);
96         }
97
98     @Override
99     public boolean equals(Object o) {
100        if (o instanceof SingleConnection) {
```

```
101         return super.equals(o);
102     }
103     return false;
104 }
105 }
106
107 public static class MultiConnection extends Connection {
108     public MultiConnection(int outIndex, int inIndex) {
109         super(outIndex, inIndex);
110     }
111
112     @Override
113     public boolean equals(Object o) {
114         if (o instanceof MultiConnection) {
115             return super.equals(o);
116         }
117         return false;
118     }
119 }
120
121 public static class SingleToMultiConnection extends Connection {
122     public SingleToMultiConnection(int outIndex, int inIndex) {
123         super(outIndex, inIndex);
124     }
125
126     @Override
127     public boolean equals(Object o) {
128         if (o instanceof SingleToMultiConnection) {
129             return super.equals(o);
130         }
131         return false;
132     }
133 }
134
135 public static class MultiToSingleConnection extends Connection {
136     public MultiToSingleConnection(int outIndex, int inIndex) {
137         super(outIndex, inIndex);
138     }
139
140     @Override
141     public boolean equals(Object o) {
142         if (o instanceof SingleToMultiConnection) {
143             return super.equals(o);
144         }
145         return false;
146     }
147 }
148
149 public List<SingleConnection> getConnections() {
150     return new ArrayList<SingleConnection>(connections);
151 }
152
153 public Type getInputType() {
154     return inputType;
155 }
```

```
156
157     public Type getOutputType() {
158         return outputType;
159     }
160
161     private boolean trivialConnect(Type output, Type input,
162             List<Connection> autoConnections) {
163         return trivialConnect(new int[]{0}, new int[]{0}, new int[]{0},
164             output, input, autoConnections);
165     }
166
167     private boolean trivialConnect(int[] outMulti, int[] outSingle, int[] inIndex,
168             Type output, Type input, List<Connection> autoConnections) {
169         while (output.getTypeCode() == TypeCode.AUTO) {
170             outMulti[0] += 1;
171             output = ((TypeAuto)output).getType();
172         }
173
174         switch (input.getTypeCode()) {
175             case AUTO:
176                 TypeAuto aut = (TypeAuto)input;
177                 if (aut.getType().getTypeCode() == TypeCode.UNKNOWN) {
178                     switch (output.getTypeCode()) {
179                         case LIST:
180                         case RECORD:
181                         case AUTO:
182                             // System.out.println("trivial connect: add multi connection "+new
183                             MultiConnection(outMulti[0], inIndex[0]));
184                             autoConnections.add(new MultiConnection(outMulti[0], inIndex[0]));
185                             outSingle[0] += getIndexCount(output);
186                             //System.out.println("increment single index with "+getIndexCount(output)+" to
187                             "+outSingle[0]);
188                             outMulti[0] += getMultiCount(output);
189                             //System.out.println("increment multi index with "+getMultiCount(output)+" to
190                             "+outMulti[0]);
191                             break;
192                         default:
193                             //System.out.println("trivial connect: add single to multi connection "+new
194                             SingleToMultiConnection(outSingle[0], inIndex[0]));
195                             autoConnections.add(new SingleToMultiConnection(outSingle[0], inIndex[0]));
196                             outSingle[0] += 1;
197                             //System.out.println("increment single index with 1 to"+outSingle[0]);
198                             break;
199                         }
200                         inIndex[0] += 1;
201                         return trivialConnect(outMulti, outSingle, inIndex,
202                             output, aut.getType(), autoConnections);
203
204             case LIST:
205                 if (output.getTypeCode() != TypeCode.LIST)
206                     return false;
207                 inIndex[0] += 1;
```

```
207         outMulti[0] += 1;
208         return trivialConnect(outMulti, outSingle, inIndex,
209             ((TypeList)output).getChildType(),
210             ((TypeList)input).getChildType(), autoConnections);
211
212     case RECORD:
213         if (output.getTypeCode() != TypeCode.RECORD)
214             return false;
215         inIndex[0] += 1;
216         outMulti[0] += 1;
217
218         TypeRecord inrec = (TypeRecord)input;
219         TypeRecord outrec = (TypeRecord)output;
220
221         Map<String, Integer> inMultiMap = new HashMap<>();
222         int index = inIndex[0];
223         for (RecordColumn c : inrec.getColumnsInOrder()) {
224             inMultiMap.put(c.getId(), index);
225             index += getMultiCount(c.getType());
226         }
227
228         Map<String, Type> incols = inrec.getColumns();
229         List<RecordColumn> outcols = outrec.getColumnsInOrder();
230         if (incols.size() != outcols.size())
231             return false;
232         for (RecordColumn c : outcols) {
233             Type inType = incols.get(c.getId());
234             if (inType == null)
235                 return false;
236             inIndex[0] = inMultiMap.get(c.getId());
237             if (!trivialConnect(outMulti, outSingle, inIndex,
238                 c.getType(), inType, autoConnections))
239                 return false;
240         }
241         return true;
242
243     default:
244         outSingle[0] += 1;
245         return output.equals(input);
246     }
247 }
248
249 public TypeConnection(Type output, Type input, boolean autoConnect) {
250     outputType = output;
251     inputType = input;
252     this.autoConnect = autoConnect;
253 }
254
255 private int getIndexCount(Type t) {
256     switch (t.getTypeCode()) {
257         case NUMBER:
258         case STRING:
259         case TIME:
260         case BOOL:
261         case UNKNOWN:
```

```
262     case ERROR:
263         return 1;
264     case AUTO:
265         return getIndexCount(((TypeAuto)t).getType());
266     case LIST:
267         return getIndexCount(((TypeList)t).getChildType());
268     case RECORD:
269         int ret = 0;
270         for (RecordColumn col : ((TypeRecord)t).getColumnsInOrder()) {
271             ret += getIndexCount(col.getType());
272         }
273         return ret;
274     default:
275         throw new RuntimeException();
276     }
277 }
278
279 private int getMultiCount(Type t) {
280     switch (t.getTypeCode()) {
281     case NUMBER:
282     case STRING:
283     case TIME:
284     case BOOL:
285     case UNKNOWN:
286     case ERROR:
287         return 0;
288     case AUTO:
289         return 1+getMultiCount(((TypeAuto)t).getType());
290     case LIST:
291         return 1+getMultiCount(((TypeList)t).getChildType());
292     case RECORD:
293         int ret = 1;
294         for (RecordColumn col : ((TypeRecord)t).getColumnsInOrder()) {
295             ret += getMultiCount(col.getType());
296         }
297         return ret;
298     default:
299         throw new RuntimeException();
300     }
301 }
302
303 private void connectAll(Type output, int[] outputIndex, Type input, int[] inputIndex) {
304     while (input.getTypeCode() == TypeCode.AUTO) {
305         input = ((TypeAuto)input).getType();
306     }
307     while (output.getTypeCode() == TypeCode.AUTO) {
308         output = ((TypeAuto)output).getType();
309     }
310     switch (output.getTypeCode()) {
311     case NUMBER:
312     case STRING:
313     case TIME:
314     case BOOL:
315     case ERROR:
316         addConnection(new SingleConnection(outputIndex[0], inputIndex[0]));

```

```
317         outputIndex[0]++;
318         inputIndex[0]++;
319         break;
320     case LIST:
321         connectAll(((TypeList)output).getChildType(), outputIndex,
322                     ((TypeList)input).getChildType(), inputIndex);
323         break;
324     case RECORD:
325         TypeRecord inputRec = (TypeRecord)input;
326         List<RecordColumn> inputCols = inputRec.getColumnsInOrder();
327         HashMap<String, Integer> inputIndices = new HashMap<>();
328         int sum = inputIndex[0];
329         for (RecordColumn col : inputCols) {
330             inputIndices.put(col.getId(), new Integer(sum));
331             sum += getIndexCount(col.getType());
332         }
333         TypeRecord outputRec = (TypeRecord)output;
334         Map<String, Type> inputMap = inputRec.getColumns();
335         List<RecordColumn> outputCols = outputRec.getColumnsInOrder();
336         for (RecordColumn col : outputCols) {
337             inputIndex[0] = inputIndices.get(col.getId());
338             connectAll(col.getType(), outputIndex, inputMap.get(col.getId()), inputIndex);
339         }
340         break;
341     default:
342         throw new RuntimeException();
343     }
344 }
345
346 public List<PathNode> getIndexPath(int index, Type portType) {
347     if (index < 0)
348         throw new IndexOutOfBoundsException("type index too small: " + index);
349     List<PathNode> ret = doGetIndexPath(new int[]{index}, portType, null, new ArrayList<>());
350     if (ret == null)
351         throw new IndexOutOfBoundsException("type index too big: " + index);
352     return ret;
353 }
354
355 private List<PathNode> doGetIndexPath(int[] index, Type portType, String colName, List<
356                                         PathNode> path) {
357     List<PathNode> ret;
358     path.add(new PathNode(colName, portType));
359
360     switch (portType.getTypeCode()) {
361     case AUTO:
362         TypeAuto portAuto = (TypeAuto)portType;
363         ret = doGetIndexPath(index, portAuto.getType(), null, path);
364         return ret;
365     case LIST:
366         TypeList portList = (TypeList)portType;
367         ret = doGetIndexPath(index, portList.getChildType(), null, path);
368 }
```

```
371         return ret;
372
373     case RECORD:
374         TypeRecord portRecord = (TypeRecord)portType;
375         List<RecordColumn> cols = portRecord.getColumnsInOrder();
376         for (RecordColumn c : cols) {
377             List<PathNode> before = new ArrayList<>(path);
378             ret = doGetIndexPath(index, c.getType(), c.getId(), path);
379             if (ret != null) {
380                 return ret;
381             }
382             path = before;
383         }
384         return null;
385
386     default:
387         if (index[0] == 0) {
388             return path;
389         } else {
390             index[0] -= 1;
391             return null;
392         }
393     }
394 }
395
396 private Type getIndexType(int index, Type portType) {
397     if (index < 0)
398         throw new IndexOutOfBoundsException("type index too small: " + index);
399     Type ret = doGetIndexType(new int[]{index}, portType);
400     if (ret == null)
401         throw new IndexOutOfBoundsException("type index too big: " + index);
402     return ret;
403 }
404
405 private Type doGetIndexType(int[] index, Type portType) {
406     switch (portType.getTypeCode()) {
407         case AUTO:
408             TypeAuto portAuto = (TypeAuto)portType;
409             return doGetIndexType(index, portAuto.getType());
410
411         case LIST:
412             TypeList portList = (TypeList)portType;
413             return doGetIndexType(index, portList.getChildType());
414
415         case RECORD:
416             TypeRecord portRecord = (TypeRecord)portType;
417             List<RecordColumn> cols = portRecord.getColumnsInOrder();
418             for (RecordColumn c : cols) {
419                 Type ret = doGetIndexType(index, c.getType());
420                 if (ret != null)
421                     return ret;
422             }
423             return null;
424
425     default:
```

```
426         if (index[0] == 0) {
427             return portType;
428         } else {
429             index[0] -= 1;
430             return null;
431         }
432     }
433 }
434
435 private String getTypeString(Type portType, int index) {
436     int[] idx = new int[]{index};
437     return doGetTypeString(portType, idx);
438 }
439
440 private String doGetTypeString(Type portType, int[] index) {
441     switch (portType.getTypeCode()) {
442     case AUTO:
443         return doGetTypeString(((TypeAuto)portType).getType(), index);
444
445     case LIST:
446         String ret = doGetTypeString(((TypeList)portType).getChildType(), index);
447         if (index[0] < 0)
448             return "List("+ret+ ")";
449         return ret;
450
451     case RECORD:
452         TypeRecord portRecord = (TypeRecord)portType;
453         List<RecordColumn> cols = portRecord.getColumnsInOrder();
454         for (RecordColumn c : cols) {
455             String ret1 = doGetTypeString(c.getType(), index);
456             if (index[0] < 0) {
457                 return ret1;
458             }
459         }
460         return "";
461
462     default:
463         index[0] -= 1;
464         if (index[0] == -1) {
465             return portType.toString();
466         }
467         return "";
468     }
469 }
470
471 private int getListNest(Type portType, int index) {
472     int[] idx = new int[]{index};
473     return doGetListNest(portType, idx);
474 }
475
476 private int doGetListNest(Type portType, int[] index) {
477     switch (portType.getTypeCode()) {
478     case AUTO:
479         return doGetListNest(((TypeAuto)portType).getType(), index);
480     }
```

```
481     case LIST:
482         int ret = doGetListNest(((TypeList)portType).getChildType(), index);
483         if (index[0] < 0)
484             return ret+1;
485         return ret;
486
487     case RECORD:
488         TypeRecord portRecord = (TypeRecord)portType;
489         List<RecordColumn> cols = portRecord.getColumnsInOrder();
490         for (RecordColumn c : cols) {
491             int ret1 = doGetListNest(c.getType(), index);
492             if (index[0] < 0) {
493                 return ret1;
494             }
495         }
496         return 0;
497
498     default:
499         index[0] -= 1;
500         return 0;
501     }
502 }
503
504 private LinkedList<TypeList> getContainingLists(Type portType, int index) {
505     int[] idx = new int[]{index};
506     return doGetContainingList(portType, idx);
507 }
508
509 private LinkedList<TypeList> doGetContainingList(Type portType, int[] index) {
510     switch (portType.getTypeCode()) {
511         case AUTO:
512             TypeAuto portAuto = (TypeAuto)portType;
513             return doGetContainingList(portAuto.getType(), index);
514
515         case LIST:
516             TypeList portList = (TypeList)portType;
517             LinkedList<TypeList> ret = doGetContainingList(portList.getChildType(), index);
518             if (index[0] < 0)
519                 ret.add(0, portList);
520             return ret;
521
522         case RECORD:
523             TypeRecord portRecord = (TypeRecord)portType;
524             List<RecordColumn> cols = portRecord.getColumnsInOrder();
525             for (RecordColumn c : cols) {
526                 LinkedList<TypeList> ret1 = doGetContainingList(c.getType(), index);
527                 if (index[0] < 0) {
528                     return ret1;
529                 }
530             }
531             return new LinkedList<>();
532
533     default:
534         index[0] -= 1;
535         return new LinkedList<>();
```

```
536     }
537 }
538
539 private Type getMultiType(int index, Type portType) {
540     if (index < 0)
541         throw new IndexOutOfBoundsException("multi type index too small: " + index);
542     Type ret = doGetMultiType(new int[]{index}, portType);
543     if (ret == null)
544         throw new IndexOutOfBoundsException("multi type index too big: " + index);
545     return ret;
546 }
547
548 private Type doGetMultiType(int[] index, Type portType) {
549     switch (portType.getTypeCode()) {
550     case AUTO:
551         TypeAuto portAuto = (TypeAuto)portType;
552         if (index[0] == 0)
553             return portAuto;
554         index[0] -= 1;
555         return doGetMultiType(index, portAuto.getType());
556
557     case LIST:
558         TypeList portList = (TypeList)portType;
559         if (index[0] == 0)
560             return portList;
561         index[0] -= 1;
562         return doGetMultiType(index, portList.getChildType());
563
564     case RECORD:
565         TypeRecord portRecord = (TypeRecord)portType;
566         if (index[0] == 0)
567             return portRecord;
568         index[0] -= 1;
569         List<RecordColumn> cols = portRecord.getColumnsInOrder();
570         for (RecordColumn c : cols) {
571             Type ret = doGetMultiType(index, c.getType());
572             if (ret != null)
573                 return ret;
574         }
575         return null;
576
577     default:
578         return null;
579     }
580 }
581
582 private int getMultiStart(int index, Type portType) {
583     if (index < 0)
584         throw new IndexOutOfBoundsException("multi type index too small: " + index);
585     int ret = doGetMultiStart(new int[]{index}, new int[]{0}, portType);
586     if (ret < 0)
587         throw new IndexOutOfBoundsException("multi type index too big: " + index);
588     return ret;
589 }
590
```

```
591     private int doGetMultiStart(int[] multiIndex, int[] singleIndex, Type portType) {
592         switch (portType.getTypeCode()) {
593             case AUTO:
594                 TypeAuto portAuto = (TypeAuto)portType;
595                 if (multiIndex[0] == 0)
596                     return singleIndex[0];
597                 multiIndex[0] -= 1;
598                 return doGetMultiStart(multiIndex, singleIndex, portAuto.getType());
599
600             case LIST:
601                 TypeList portList = (TypeList)portType;
602                 if (multiIndex[0] == 0)
603                     return singleIndex[0];
604                 multiIndex[0] -= 1;
605                 return doGetMultiStart(multiIndex, singleIndex, portList.getChildType());
606
607             case RECORD:
608                 TypeRecord portRecord = (TypeRecord)portType;
609                 if (multiIndex[0] == 0)
610                     return singleIndex[0];
611                 multiIndex[0] -= 1;
612                 List<RecordColumn> cols = portRecord.getColumnsInOrder();
613                 for (RecordColumn c : cols) {
614                     int ret = doGetMultiStart(multiIndex, singleIndex, c.getType());
615                     if (ret >= 0)
616                         return ret;
617                 }
618                 return -1;
619
620             default:
621                 singleIndex[0] += 1;
622                 return -1;
623         }
624     }
625
626     private Type deduceAutoType(int inIndex, Type deduceType, Type[] deduceOut, Type portType) {
627         return doDeduceAutoType(new int[]{inIndex}, portType, deduceType, deduceOut);
628     }
629
630     private Type doDeduceAutoType(int[] index, Type inPort, Type deduceType, Type[] deduceOut) {
631         switch (inPort.getTypeCode()) {
632             case AUTO:
633                 TypeAuto portAuto = (TypeAuto)inPort;
634                 if (index[0] == 0) {
635                     deduceOut[0] = new TypeAuto(deduceType);
636                     index[0] -= 1;
637                     return deduceOut[0];
638                 }
639                 index[0] -= 1;
640                 return new TypeAuto(doDeduceAutoType(index, portAuto.getType(), deduceType, deduceOut));
641             case LIST:
642                 TypeList portList = (TypeList)inPort;
643                 index[0] -= 1;
```

```
645         return new TypeList(doDeduceAutoType(index, portList.getChildType(), deduceType,
646                                         deduceOut));
647
648     case RECORD:
649         TypeRecord portRecord = (TypeRecord)inPort;
650         index[0] -= 1;
651         List<RecordColumn> cols = portRecord.getColumnsInOrder();
652         List<RecordColumn> ncols = new ArrayList<>();
653         for (RecordColumn c : cols) {
654             Type t = doDeduceAutoType(index, c.getType(), deduceType, deduceOut);
655             ncols.add(new RecordColumn(c.getId(), t));
656         }
657         return new TypeRecord(ncols);
658
659     default:
660         return inPort;
661     }
662
663     public void addMultiConnection(MultiConnection conn) {
664         Type out = getMultiType(conn.outputIndex, outputType);
665         Type in = getMultiType(conn.inputIndex, inputType);
666         int multi = conn.outputIndex;
667         int single = getMultiStart(conn.outputIndex, outputType);
668         addMultiInputConnection(out, in, multi, single, conn.inputIndex);
669     }
670
671     private void addMultiInputConnection(Type out, Type in, int outMulti,
672                                         int outSingle, int inMulti) {
673         //System.out.println("add multi connection in type "+in);
674         List<Connection> autoConnections = new ArrayList<>();
675         if (!trivialConnect(new int[]{outMulti}, new int[]{outSingle},
676                             new int[]{inMulti}, out, in, autoConnections) ||
677             !out.isConcrete(false) || !in.isConcrete(true)) {
678             throw new IllegalArgumentException("Type mismatch");
679         }
680
681         out = removeAutoTypes(out);
682
683         Type origType = inputType;
684         Type newType = inputType;
685         for (int i = autoConnections.size()-1; i >= 0; i--) {
686             Connection conn = autoConnections.get(i);
687
688             //System.out.println("auto connection "+conn);
689
690             int single = getMultiStart(conn.inputIndex, inputType);
691             int multi = conn.inputIndex;
692             Type outType;
693             if (conn instanceof MultiConnection) {
694                 //System.out.println("multi connection");
695                 outType = getMultiType(conn.outputIndex, outputType);
696             } else {
697                 //System.out.println("single connection");
698                 outType = getIndexType(conn.outputIndex, outputType);
```

```
699         }
700         outType = removeAutoTypes(outType);
701
702         //System.out.println("Auto connectin output type "+outType);
703
704         Type[] deduced = new Type[1];
705         newType = deduceAutoType(multi, outType, deduced, newType);
706         //System.out.println("new type "+newType);
707         int count = getIndexCount(in);
708         if (!hasInputConnectionInRange(single, single+count)) {
709             updateConnections(single, getIndexCount(deduced[0])-1);
710         }
711     }
712     if (autoConnections.size() > 0) {
713         if (autoListener != null) {
714             try {
715                 autoListener.autoTypeInputChanged(newType);
716             } catch (TypeAutoChangeException ex) {
717                 throw new IllegalArgumentException(ex.getMessage());
718             }
719         }
720         inputType = newType;
721         in = getMultiType(inMulti, inputType);
722     }
723
724     //System.out.println("output type "+outputType);
725     //System.out.println("input type "+inputType);
726
727     int inSingle = getMultiStart(inMulti, inputType);
728     int origSize = connections.size();
729     try {
730         connectAll(out, new int[]{outSingle}, in, new int[]{inSingle});
731     } catch (IllegalArgumentException ex) {
732         while (connections.size() > origSize)
733             removeConnection(connections.get(connections.size()-1));
734         inputType = origType;
735         throw ex;
736     }
737 }
738
739 private Type removeAutoTypes(Type t) {
740     switch (t.getTypeCode()) {
741     case AUTO:
742         TypeAuto a = (TypeAuto)t;
743         return removeAutoTypes(a.getType());
744
745     case LIST:
746         TypeList lst = (TypeList)t;
747         Type child = removeAutoTypes(lst.getChildType());
748         return new TypeList(child);
749
750     case RECORD:
751         TypeRecord rec = (TypeRecord)t;
752         List<RecordColumn> cols = rec.getColumnsInOrder();
753         List<RecordColumn> ncols = new ArrayList<>();
```

```
754         for (RecordColumn c : cols) {
755             Type nc = removeAutoTypes(c.getType());
756             ncols.add(new RecordColumn(c.getId(), nc));
757         }
758         return new TypeRecord(ncols);
759     }
760     default:
761         return t;
762     }
763 }
764
765 public void addMultiToSingleConnection(MultiToSingleConnection conn) {
766     Type out = getMultiType(conn.outputIndex, outputType);
767     while (out.getTypeCode() == TypeCode.AUTO) {
768         out = ((TypeAuto)out).getType();
769     }
770     switch (out.getTypeCode()) {
771     case LIST:
772     case RECORD:
773         throw new IllegalArgumentException("Type mismatch");
774     default:
775         break;
776     }
777     int single = getMultiStart(conn.outputIndex, outputType);
778     addConnection(new SingleConnection(single, conn.inputIndex));
779 }
780
781 public void addSingleToMultiConnection(SingleToMultiConnection conn) {
782     Type out = getIndexType(conn.outputIndex, outputType);
783     Type in = getMultiType(conn.inputIndex, inputType);
784     int multi = getMultiOfIndex(conn.outputIndex, outputType);
785     int single = conn.outputIndex;
786     addMultiInputConnection(out, in, multi, single, conn.inputIndex);
787 }
788
789 private void updateConnections(int after, int increment) {
790     //System.out.println("update after index "+after);
791     //System.out.println("update increment "+increment);
792     ArrayList<SingleConnection> newConns = new ArrayList<>();
793     for (SingleConnection conn : connections) {
794         if (conn.inputIndex <= after) {
795             newConns.add(conn);
796         } else {
797             //System.out.println("update connection "+conn+" to "+new SingleConnection(conn.
798             outputIndex, conn.inputIndex+increment));
799             newConns.add(new SingleConnection(conn.outputIndex, conn.inputIndex+increment));
800         }
801     }
802     connections = newConns;
803 }
804
805 private int getMultiOfIndex(int index, Type portType) {
806     return getMultiOfIndex(new int[]{0}, new int[]{0}, index, portType);
807 }
808
809 private int getMultiOfIndex(int[] multi, int[] index, int search, Type portType) {
```

```
808     switch (portType.getTypeCode()) {
809     case AUTO:
810         int start = index[0];
811         int end = getIndexCount(portType) + start;
812         if (search >= start && search < end)
813             return multi[0];
814         multi[0] += 1;
815         return getListNest(multi, index, search, ((TypeAuto)portType).getType());
816
817     case LIST:
818         int start1 = index[0];
819         int end1 = getIndexCount(portType) + start1;
820         if (search >= start1 && search < end1)
821             return multi[0];
822         multi[0] += 1;
823         return getListNest(multi, index, search, ((TypeList)portType).getChildType());
824
825     case RECORD:
826         int start2 = index[0];
827         int end2 = getIndexCount(portType) + start2;
828         if (search >= start2 && search < end2)
829             return multi[0];
830         multi[0] += 1;
831         TypeRecord rec = (TypeRecord)portType;
832         for (RecordColumn c : rec.getColumnsInOrder()) {
833             int r = getListNest(multi, index, search, c.getType());
834             if (r != -1)
835                 return r;
836         }
837         return -1;
838
839     default:
840         index[0] -= 1;
841         return -1;
842     }
843 }
844
845 public void addConnection(SingleConnection conn) {
846     Type input = getIndexType(conn.inputIndex, inputType);
847     Type output = getIndexType(conn.outputIndex, outputType);
848
849     if (!input.isConcrete(false) || !output.isConcrete(false)) {
850         throw new IllegalArgumentException("Cannot connect unknown type");
851     }
852
853     for (SingleConnection c : connections) {
854         if (c.inputIndex == conn.inputIndex) {
855             throw new IllegalArgumentException("Multiple input connections");
856         }
857     }
858
859     if (getListNest(inputType, conn.inputIndex) != getListNest(outputType, conn.outputIndex)
860     ||
861         !input.equals(output)) {
862         throw new IllegalArgumentException(

```

```
862             "Types " + getTypeString(outputType, conn.outputIndex) + " and " +
863             getTypeString(inputType, conn.inputIndex) + " are incompatible");
864         }
865
866         LinkedList<TypeList> containingInputLists = getContainingLists(inputType, conn.inputIndex)
867         ;
868         LinkedList<TypeList> containingOutputLists = getContainingLists(outputType, conn.
869         outputIndex);
870         for (SingleConnection c : connections) {
871             LinkedList<TypeList> ilists = getContainingLists(inputType, c.inputIndex);
872             int sameParentCount = 0;
873             int size = Math.min(ilists.size(), containingInputLists.size());
874             for (int i = 0; i < size; i++) {
875                 if (ilists.get(i) == containingInputLists.get(i)) {
876                     sameParentCount += 1;
877                 }
878             }
879             LinkedList<TypeList> olists = getContainingLists(outputType, c.outputIndex);
880             for (int i = 0; i < sameParentCount; i++)
881                 if (olists.get(i) != containingOutputLists.get(i))
882                     throw new IllegalArgumentException("Input list connected with distinct output
883 lists");
884         }
885
886         //System.out.println("ADD CONNECTION "+conn);
887         connections.add(conn);
888     }
889
890     private boolean collapseAutoType(int index) {
891         int[] start = new int[1];
892         start[0] = -1;
893         int[] size = new int[1];
894         size[0] = -1;
895         inputType = doCollapseAutoType(new int[]{0}, new int[]{0}, index, inputType, start, size);
896         if (start[0] >= 0) {
897             if (autoListener != null) {
898                 try {
899                     autoListener.autoTypeInputChanged(inputType);
900                 } catch (TypeAutoChangeException e) {
901                     throw new RuntimeException(e);
902                 }
903             }
904         }
905         return true;
906     }
907
908     private Type doCollapseAutoType(int[] multiIndex, int[] currIndex, int searchIndex, Type
909     portType, int[] startOut, int[] sizeOut) {
910         switch (portType.getTypeCode()) {
911         case AUTO:
912             int start = currIndex[0];
913             TypeAuto portAuto = (TypeAuto)portType;
```

```
913         int count = getIndexCount(portAuto);
914         if (start <= searchIndex &&
915             start+count > searchIndex &&
916             !hasInputConnectionInRange(start, start+count) &&
917             autoListener.autoTypeInputMayChange()) {
918             startOut[0] = start;
919             //System.out.println("auto collapse start "+start);
920             sizeOut[0] = count;
921             //System.out.println("auto collapse size "+count);
922             multiIndex[0] += getMultiCount(portType);
923             currIndex[0] += getIndexCount(portType);
924             return new TypeAuto();
925         }
926         multiIndex[0] += 1;
927         return new TypeAuto(doCollapseAutoType(multiIndex, currIndex, searchIndex, portAuto.
928             getType(), startOut, sizeOut));
929     case LIST:
930         TypeList portList = (TypeList)portType;
931         multiIndex[0] += 1;
932         return new TypeList(doCollapseAutoType(multiIndex, currIndex, searchIndex, portList.
933             getChildType(), startOut, sizeOut));
934     case RECORD:
935         TypeRecord portRecord = (TypeRecord)portType;
936         multiIndex[0] += 1;
937         List<RecordColumn> cols = portRecord.getColumnsInOrder();
938         List<RecordColumn> ncols = new ArrayList<>();
939         for (RecordColumn c : cols) {
940             Type r = doCollapseAutoType(multiIndex, currIndex, searchIndex, c.getType(),
941             startOut, sizeOut);
942             ncols.add(new RecordColumn(c.getId(), r));
943         }
944         return new TypeRecord(ncols);
945     default:
946         currIndex[0] += 1;
947         return portType;
948     }
949 }
950
951 public boolean removeConnection(SingleConnection conn) {
952     //System.out.println("REMOVE CONNECTION "+conn);
953     int size = connections.size();
954     for (int i = 0; i < size; i++) {
955         SingleConnection c2 = connections.get(i);
956         if (c2.equals(conn)) {
957             connections.remove(i);
958             return collapseAutoType(conn.inputIndex);
959         }
960     }
961     throw new IllegalArgumentException("connection " + conn + " does not exist");
962 }
963
964 public boolean isConnected() {
```

```
965     ArrayList<Integer> inputCons = new ArrayList<>();
966     for (SingleConnection c : connections)
967         inputCons.add(new Integer(c.inputIndex));
968     Collections.sort(inputCons);
969     Integer prev = -1;
970     for (Integer curr : inputCons) {
971         if (curr != prev+1)
972             return false;
973         prev = curr;
974     }
975
976     try {
977         getIndexType(inputCons.size(), inputType);
978     } catch (IndexOutOfBoundsException e) {
979         return true;
980     }
981     return false;
982 }
983
984 private boolean hasInputConnectionInRange(int start, int end) {
985     for (SingleConnection c : connections) {
986         if (c.inputIndex >= start && c.inputIndex < end) {
987             return true;
988         }
989     }
990     return false;
991 }
992
993 public boolean hasConnection(int outIndex, int inIndex) {
994     for (SingleConnection c : connections) {
995         if (c.inputIndex == inIndex && c.outputIndex == outIndex)
996             return true;
997     }
998     return false;
999 }
1000
1001 public void setTypeAutoListener(TypeAutoInputListener autoListener) {
1002     this.autoListener = autoListener;
1003 }
1004
1005 public void autoConnect() {
1006     try {
1007         doAutoConnect();
1008     } catch (IllegalArgumentException e) {}
1009 }
1010
1011 private void doAutoConnect() {
1012     Type output = outputType;
1013     Type input = inputType;
1014
1015     List<Connection> conns = new ArrayList<>();
1016     if (autoConnect && trivialConnect(output, input, conns) &&
1017         output.isConcrete(false) && input.isConcrete(true)) {
1018         switch (output.getTypeCode()) {
1019             case LIST:
```

```

1020         case AUTO:
1021         case RECORD:
1022             switch (inputType.getTypeCode()) {
1023                 case LIST:
1024                     case AUTO:
1025                     case RECORD:
1026                         addMultiConnection(new MultiConnection(0, 0));
1027                         break;
1028                     default:
1029                         addMultiToSingleConnection(new MultiToSingleConnection(0, 0));
1030                         break;
1031                 }
1032                 break;
1033             default:
1034                 switch (inputType.getTypeCode()) {
1035                     case LIST:
1036                     case AUTO:
1037                     case RECORD:
1038                         addSingleToMultiConnection(new SingleToMultiConnection(0, 0));
1039                         break;
1040                     default:
1041                         addConnection(new SingleConnection(0, 0));
1042                         break;
1043                 }
1044                 break;
1045             }
1046         }
1047     autoConnect = false;
1048 }
1049 }
1050 }
```

LISTING E.64: lang/type/TypeConnection.java

```

1 package lang.type;
2
3 public class TypeBool implements Type {
4     @Override
5     public boolean hasAuto() {
6         return false;
7     }
8
9     @Override
10    public boolean isConcrete(boolean b) {
11        return true;
12    }
13
14    @Override
15    public boolean equals(Object oth) {
16        if (oth instanceof TypeBool)
17            return true;
18        return defaultEquals(oth);
19    }
20
21    @Override
```

```

22     public int hashCode() {
23         return TypeCode.BOOL.hashCode();
24     }
25
26     @Override
27     public String toString() {
28         return "Boolean";
29     }
30
31     @Override
32     public String getShortDescription() {
33         return toString();
34     }
35
36     @Override
37     public TypeCode getTypeCode() {
38         return TypeCode.BOOL;
39     }
40 }
```

LISTING E.65: lang/type/TypeBool.java

```

1 package lang.type;
2
3 public interface TypeAutoInputListener {
4     void autoTypeInputChanged(Type newType) throws TypeAutoChangeException;
5     void autoTypeInputRemoved();
6     boolean autoTypeInputMayChange();
7 }
```

LISTING E.66: lang/type/TypeAutoInputListener.java

```

1 package lang.type;
2
3 public class TypeString implements Type {
4     @Override
5     public boolean hasAuto() {
6         return false;
7     }
8
9     @Override
10    public boolean isConcrete(boolean b) {
11        return true;
12    }
13
14    @Override
15    public boolean equals(Object oth) {
16        if (oth instanceof TypeString)
17            return true;
18        return defaultEquals(oth);
19    }
20
21    @Override
22    public int hashCode() {
```

```
23         return TypeCode.STRING.hashCode();
24     }
25
26     @Override
27     public String toString() {
28         return "String";
29     }
30
31     @Override
32     public TypeCode getTypeCode() {
33         return TypeCode.STRING;
34     }
35
36     @Override
37     public String getShortDescription() {
38         return toString();
39     }
40 }
```

LISTING E.67: lang/type/TypeString.java

```
1 package lang.type;
2
3 public class TypeNumber implements Type {
4     @Override
5     public boolean hasAuto() {
6         return false;
7     }
8
9     @Override
10    public boolean isConcrete(boolean b) {
11        return true;
12    }
13
14    @Override
15    public boolean equals(Object oth) {
16        if (oth instanceof TypeNumber)
17            return true;
18        return defaultEquals(oth);
19    }
20
21    @Override
22    public int hashCode() {
23        return TypeCode.NUMBER.hashCode();
24    }
25
26    @Override
27    public String toString() {
28        return "Number";
29    }
30
31    @Override
32    public String getShortDescription() {
33        return toString();
34    }
}
```

```

35
36     @Override
37     public TypeCode getTypeCode() {
38         return TypeCode.NUMBER;
39     }
40 }
```

LISTING E.68: lang/type/TypeNumber.java

```

1 package gui;
2
3 import java.awt.BorderLayout;
4 import java.awt.Graphics;
5
6 import javax.swing.JPanel;
7 import javax.swing.JSplitPane;
8
9 import util.UtilScrollPane;
10 import lang.constant.Constant;
11 import lang.constant.InvalidConstantException;
12
13 @SuppressWarnings("serial")
14 public class ConstantDefiner extends JPanel {
15     private TypeSelector typeSelector;
16     private ValuePanel valuePanel;
17
18     public ConstantDefiner(Graphics g) {
19         super(new BorderLayout());
20         typeSelector = new TypeSelector("Select Type");
21         valuePanel = new ValuePanel(this, g);
22         typeSelector.addTypeChangedListener(valuePanel);
23         UtilScrollPane typeScrollPane = new UtilScrollPane(typeSelector);
24         UtilScrollPane valueScrollPane = new UtilScrollPane(valuePanel);
25         JSplitPane splitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
26             typeScrollPane, valueScrollPane);
27         splitPane.setDividerLocation(0.25);
28         splitPane.setResizeWeight(0.25);
29         add(splitPane, BorderLayout.CENTER);
30     }
31
32     public TypeSelector getTypeSelector() {
33         return typeSelector;
34     }
35
36     public Constant getValue() throws InvalidConstantException {
37         return valuePanel.getValue();
38     }
39 }
```

LISTING E.69: gui/ConstantDefiner.java

```

1 package gui;
2
3 import java.awt.Point;
```

```
4 import java.io.PrintStream;
5 import java.util.ArrayList;
6 import java.util.HashMap;
7 import java.util.List;
8 import java.util.Map;
9
10 import gui.controlflow.ControlArrow;
11 import gui.controlflow.ControlPart;
12 import gui.controlflow.ControlPartTypeConnection;
13 import gui.dataflow.DataPart;
14 import gui.dataflow.DataPartArithmetic;
15 import gui.dataflow.DataPartCast;
16 import gui.dataflow.DataPartComparison;
17 import gui.dataflow.DataPartConstant;
18 import gui.dataflow.DataPartFunctional;
19 import gui.dataflow.DataPartListOperation;
20 import gui.dataflow.DataPartParameter;
21 import gui.dataflow.DataPartRecordConstructor;
22 import gui.dataflow.DataPartResult;
23 import gui.dataflow.DataPartSink;
24 import gui.dataflow.DataPartStringFormat;
25 import gui.dataflow.DataPartStringOperation;
26 import gui.dataflow.DataPartSubFlow;
27 import lang.ast.ASTCF;
28 import lang.ast.ASTCFComponent;
29 import lang.ast.ASTCFComponentDF;
30 import lang.ast.ASTCFComponentFail;
31 import lang.ast.ASTCFComponentIf;
32 import lang.ast.ASTCFComponentInteraction;
33 import lang.ast.ASTCFComponentStart;
34 import lang.ast.ASTCFComponentStop;
35 import lang.ast.ASTCFEdge;
36 import lang.ast.ASTCFTypeConnection;
37 import lang.ast.ASTDF;
38 import lang.ast.ASTDFComponent;
39 import lang.ast.ASTDFComponentArithmetic;
40 import lang.ast.ASTDFComponentCast;
41 import lang.ast.ASTDFComponentComparison;
42 import lang.ast.ASTDFComponentConstant;
43 import lang.ast.ASTDFComponentDF;
44 import lang.ast.ASTDFComponentFunctional;
45 import lang.ast.ASTDFComponentInput;
46 import lang.ast.ASTDFComponentListOperation;
47 import lang.ast.ASTDFComponentOutput;
48 import lang.ast.ASTDFComponentRecordConstructor;
49 import lang.ast.ASTDFComponentSink;
50 import lang.ast.ASTDFComponentStringFormat;
51 import lang.ast.ASTDFComponentStringOperation;
52 import lang.ast.ASTDFAEdge;
53 import lang.ast.ASTVisitorPrint;
54 import lang.type.RecordColumn;
55 import lang.type.TypeConnection;
56
57 public class AST {
58     private ASTCF rootNode;
```

```
59
60     public AST(ControlFlowPanel root) {
61         rootNode = getASTCF(root);
62     }
63
64     public void print(PrintStream out) {
65         rootNode.accept(new ASTVisitorPrint(out));
66     }
67
68     public void print() {
69         rootNode.accept(new ASTVisitorPrint(System.out));
70     }
71
72     public ASTCF getRoot() {
73         return rootNode;
74     }
75
76     private ASTCF getASTCF(ControlFlowPanel root) {
77         Map<ControlPart,ASTCFComponent> componentMap = new HashMap<>();
78         ASTCF cf = new ASTCF(root.getTitle(), root.getPlatform(), root.getUserGroup());
79         for (ControlPart p : root.getControlParts()) {
80             cf.addComponent(getASTCFComponent(p, componentMap));
81         }
82         for (ControlArrow a : root.getControlArrows()) {
83             cf.addEdge(getASTCFEdge(a, componentMap));
84         }
85         return cf;
86     }
87
88     private Map<Integer,String> getControlPartEventMap(ControlPart part, boolean hasNamedEvents) {
89         Map<Integer,String> ret = new HashMap<>();
90         Integer index = 0;
91         for (String evt : part.getEvents(true)) {
92             if (!hasNamedEvents && !evt.equals(ControlPart.errorEvent))
93                 evt = "";
94             ret.put(index, evt);
95             ret.put(index+1, evt);
96             index += 2;
97         }
98         return ret;
99     }
100
101    private ASTCFComponent getASTCFComponent(ControlPart p, Map<ControlPart,ASTCFComponent>
102                                              componentMap) {
103        ASTCFComponent ret;
104        switch (p.getType()) {
105            case DATAFLOW:
106                DataFlowPanel dfp = (DataFlowPanel)p.getTabbedPanel();
107                List<ASTDFComponent> components = new ArrayList<>();
108                Map<DataPart,ASTDFComponent> map = new HashMap<>();
109                for (DataPart dataPart : dfp.getDataParts()) {
110                    components.add(getASTDFComponent(dataPart, map));
111                }
112                ASTCFComponentDF df = new ASTCFComponentDF(p.getTitle(), p.getLocation(), p.getInputs()
113                                                       (),
```

```
112             p.getOutputs(false), components, getControlPartEventMap(p, false), p.
113             getSequenceNumber(), p.canFail());
114             for (DataPart dataPart : dfp.getDataParts()) {
115                 df.addAllDataFlowEdges(getASTDFEdges(dataPart, map));
116             }
117             ret = df;
118             break;
119         case IF:
120             ret = new ASTCFComponentIf(p.getLocation(), p.getInputs(), getControlPartEventMap(p,
121             true), p.getSequenceNumber());
122             break;
123         case START:
124             ret = new ASTCFComponentStart(p.getLocation(), p.getInputs(), getControlPartEventMap(p
125             , false), p.getSequenceNumber());
126             break;
127         case STOP:
128             ret = new ASTCFComponentStop(p.getLocation(), p.getInputs(), getControlPartEventMap(p,
129             false), p.getSequenceNumber());
130             break;
131         case FAIL:
132             ret = new ASTCFComponentFail(p.getLocation(), p.getInputs(), getControlPartEventMap(p,
133             false), p.getSequenceNumber());
134             break;
135         case INTERACTION:
136             ret = new ASTCFComponentInteraction(p.getLocation(), p.getTitle(), p.
137             getInteractionPlatform(), p.getInputs(),
138             p.getOutputs(false), getControlPartEventMap(p, true), p.getSequenceNumber());
139             break;
140         default:
141             throw new RuntimeException();
142     }
143     componentMap.put(p, ret);
144     return ret;
145 }
146
147 private List<String> getOutputList(DataPart p) {
148     List<String> ret = new ArrayList<>();
149     for (DataPart.OutputArrow a : p.getOutputArrows()) {
150         ret.add(p.getOutputName(a));
151     }
152     return ret;
153 }
154
155 private List<String> getInputList(DataPart p) {
156     List<String> ret = new ArrayList<>();
157     for (DataPart.InputArrow a : p.getInputArrows()) {
158         ret.add(p.getInputName(a));
159     }
156
157     private ASTDFComponent getASTDFComponent(DataPart dataPart, Map<DataPart,ASTDFComponent> map)
158     {
159         ASTDFComponent ret;
160         Point location = dataPart.getPointBoxLocation();
```

```
160     if (dataPart instanceof DataPartParameter) {
161         DataPartParameter p = (DataPartParameter)dataPart;
162         ret = new ASTDFComponentInput(location, p.getOriginalType(), p.getType(), p.getName(),
163             getInputList(dataPart), getListOutputList(dataPart), dataPart.getSequenceNumber());
164     } else if (dataPart instanceof DataPartResult) {
165         DataPartResult r = (DataPartResult)dataPart;
166         ret = new ASTDFComponentOutput(location, r.getOriginalType(), r.getType(), r.getName(),
167             getInputList(dataPart), getListOutputList(dataPart), dataPart.getSequenceNumber());
168     } else if (dataPart instanceof DataPartSink) {
169         DataPartSink s = (DataPartSink)dataPart;
170         ret = new ASTDFComponentSink(s.getPointBoxLocation(), s.getType(), getInputList(
171             dataPart), getListOutputList(dataPart), s.getSequenceNumber());
172     } else if (dataPart instanceof DataPartArithmetic) {
173         ret = new ASTDFComponentArithmetic(location, ((DataPartArithmetic)dataPart).getType().
174             toString(), dataPart.getShortTitle(), getInputList(dataPart), getListOutputList(dataPart),
175             dataPart.getSequenceNumber());
176     } else if (dataPart instanceof DataPartStringOperation) {
177         ret = new ASTDFComponentStringOperation(location, ((DataPartStringOperation)dataPart).
178             getStringOperationType().toString(), dataPart.getShortTitle(), getInputList(dataPart),
179             getListOutputList(dataPart), dataPart.getSequenceNumber());
180     } else if (dataPart instanceof DataPartCast) {
181         DataPartCast cast = (DataPartCast)dataPart;
182         ret = new ASTDFComponentCast(location, cast.getOriginalFromType(), cast.getFromType(),
183             cast.getToType(), dataPart.getShortTitle(), getInputList(dataPart), getListOutputList(dataPart),
184             dataPart.getSequenceNumber());
185     } else if (dataPart instanceof DataPartComparison) {
186         DataPartComparison comp = (DataPartComparison)dataPart;
187         ret = new ASTDFComponentComparison(location, comp.getComparisonType().toString(), comp.
188             getDataType(), comp.getOriginalDataType(), dataPart.getShortTitle(), getInputList(dataPart),
189             getListOutputList(dataPart), dataPart.getSequenceNumber());
190     } else if (dataPart instanceof DataPartConstant) {
191         DataPartConstant con = (DataPartConstant)dataPart;
192         ret = new ASTDFComponentConstant(location, con.getConstant(), dataPart.getShortTitle(),
193             getInputList(dataPart), getListOutputList(dataPart), dataPart.getSequenceNumber());
194     } else if (dataPart instanceof DataPartStringFormat) {
195         DataPartStringFormat f = (DataPartStringFormat)dataPart;
196         ret = new ASTDFComponentStringFormat(f.getFormat(), location, dataPart.getShortTitle(),
197             getInputList(dataPart), getListOutputList(dataPart), dataPart.getSequenceNumber());
198     } else if (dataPart instanceof DataPartRecordConstructor) {
199         DataPartRecordConstructor con = (DataPartRecordConstructor)dataPart;
200         ret = new ASTDFComponentRecordConstructor(location, con.getRecordType(), dataPart.
201             getShortTitle(), getInputList(dataPart), getListOutputList(dataPart), dataPart.getSequenceNumber(
202                ()));
203     } else if (dataPart instanceof DataPartListOperation) {
204         DataPartListOperation op = (DataPartListOperation)dataPart;
205         ret = new ASTDFComponentListOperation(location, op.getListOperationType().toString(),
206             op.getOriginalOperationTypes(), op.getOperationTypes(), dataPart.getShortTitle(),
207             getInputList(dataPart), getListOutputList(dataPart), dataPart.getSequenceNumber());
208     } else if (dataPart instanceof DataPartSubFlow) {
209         DataPartSubFlow sub = (DataPartSubFlow)dataPart;
210         DataFlowPanel subFlow = sub.getSubDataFlowPanel();
211         ret = new ASTDFComponentDF(location, getASTDF(subFlow, subFlow getArguments(), subFlow.
212             getResults(), false), getInputList(dataPart), getListOutputList(dataPart), dataPart.
213             getSequenceNumber());
214     } else if (dataPart instanceof DataPartFunctional) {
```

```
196     DataPartFunctional fun = (DataPartFunctional)dataPart;
197     DataFlowPanel subFlow = fun.getSubDataFlowPanel();
198     List<RecordColumn> inputs = subFlow getArguments();
199     List<RecordColumn> outputs = subFlow.getResults();
200     ASTDF df = getASTDF(fun.getSubDataFlowPanel(), inputs, outputs, true);
201     ret = new ASTDFComponentFunctional(location, fun.getFunctionalType().toString(),
202     inputs, outputs, df, dataPart.getShortTitle(), getInputList(dataPart), getList(dataPart),
203     dataPart.getSequenceNumber());
204     } else {
205         throw new RuntimeException();
206     }
207     map.put(dataPart, ret);
208     return ret;
209 }
210
211 private ASTDF getASTDF(DataFlowPanel dfp, List<RecordColumn> inputs, List<RecordColumn>
212     outputs, boolean fixedArgOrder) {
213     ASTDF df = new ASTDF(dfp.getTitle(), inputs, outputs, fixedArgOrder);
214     Map<DataPart,ASTDFComponent> map = new HashMap<>();
215     for (DataPart dataPart : dfp.getDataParts()) {
216         df.addComponent(getASTDFComponent(dataPart, map));
217     }
218     for (DataPart dataPart : dfp.getDataParts()) {
219         df.addAllDataFlowEdges(getASTDFEdges(dataPart, map));
220     }
221     return df;
222 }
223
224 private List<ASTDFEdge> getASTDFEdges(DataPart dataPart, Map<DataPart,ASTDFComponent> map) {
225     List<ASTDFEdge> es = new ArrayList<>();
226     for (DataPart.OutputArrow arrow : dataPart.getOutputArrows()) {
227         for (DataPart.InputArrow inputArrow : arrow.getConnections()) {
228             es.add(getASTDFEdge(arrow, inputArrow, map));
229         }
230     }
231     return es;
232 }
233
234 private ASTDFEdge getASTDFEdge(DataPart.OutputArrow out, DataPart.InputArrow in, Map<DataPart,
235     ASTDFComponent> map) {
236     DataPart outputPart = out.getDataPart();
237     ASTDFComponent output = map.get(outputPart);
238     String outputName = outputPart.getOutputName(out);
239     DataPart inputPart = in.getDataPart();
240     ASTDFComponent input = map.get(inputPart);
241     String inputName = inputPart.getInputName(in);
242     TypeConnection conn = in.getTypeConnection();
243     return new ASTDFEdge(outputName, output, inputName, input, conn);
244 }
245
246 private ASTCFEdge getASTCFEdge(ControlArrow arrow, Map<ControlPart,ASTCFComponent>
247     componentMap) {
248     int index = arrow.getStartCircleIndex();
249     ASTCFComponent start = componentMap.get(arrow.getStartControlPart());
250     ASTCFComponent end = componentMap.get(arrow.getEndControlPart());
```

Java Bank Integrated Development Environment

```
246     List<ASTCFTypeConnection> conns = new ArrayList<>();
247     for (ControlPartTypeConnection c : arrow.getArgumentConnections().values()) {
248         ASTCFComponent out = componentMap.get(c.getOutputPart());
249         ASTCFComponent in = componentMap.get(c.getInputPart());
250         conns.add(new ASTCFTypeConnection(out, c.getOutputName(), in, c.getInputName(), c));
251     }
252     return new ASTCFEdge(index, start, end, conns);
253 }
254 }
```

LISTING E.70: gui/AST.java

```
1 package gui;
2
3 import javax.swing.JPopupMenu;
4 import javax.swing.UIManager;
5
6 import util.UtilFont;
7
8 public class Main {
9     public static void main(String[] args) {
10         try {
11             UIManager.setLookAndFeel(UIManager
12                 .getCrossPlatformLookAndFeelClassName());
13         } catch (Exception e) {
14         }
15         JPopupMenu.setDefaultLightWeightPopupEnabled(false);
16         UIManager.put("OptionPane.font", UtilFont.textFont);
17         UIManager.put("OptionPane.messageFont", UtilFont.textFont);
18         UIManager.put("OptionPane.buttonFont", UtilFont.textFont);
19         new MainWindow();
20     }
21 }
```

LISTING E.71: gui/Main.java

```
1 package gui;
2
3 import java.awt.BorderLayout;
4 import java.awt.Color;
5 import java.awt.FlowLayout;
6 import java.awt.GridBagConstraints;
7 import java.awt.GridBagLayout;
8 import java.awt.Window;
9 import java.util.List;
10
11 import javax.swing.BorderFactory;
12 import javax.swing.JDialog;
13 import javax.swing.JPanel;
14 import javax.swing.JTextArea;
15 import javax.swing.border.TitledBorder;
16 import javax.swing.text.DefaultCaret;
17
18 import lang.type.RecordColumn;
```

```
19 import lang.type.Type;
20 import lang.type.TypeAuto;
21 import lang.type.TypeBool;
22 import lang.type.TypeError;
23 import lang.type.TypeList;
24 import lang.type.TypeNumber;
25 import lang.type.TypeRecord;
26 import lang.type.TypeString;
27 import lang.type.TypeTime;
28 import lang.type.TypeUnknown;
29 import util.UtilFont;
30
31 @SuppressWarnings("serial")
32 public class ShowTypePanel extends JPanel {
33
34     public ShowTypePanel(Type type, boolean isOutputType) {
35         setLayout(new FlowLayout(FlowLayout.LEADING, 0, 0));
36         JPanel panel = new JPanel(new BorderLayout());
37         panel.add(getTypeGraphics(type));
38
39         if (isOutputType)
40             panel.setBorder(BorderFactory.createTitledBorder(BorderFactory.createEmptyBorder(),
41                     "Output", TitledBorder.CENTER, TitledBorder.TOP));
42         else
43             panel.setBorder(BorderFactory.createTitledBorder(BorderFactory.createEmptyBorder(),
44                     "Input", TitledBorder.CENTER, TitledBorder.BOTTOM));
45
46         add(panel);
47     }
48
49     public ShowTypePanel(Type type) {
50         setLayout(new FlowLayout(FlowLayout.LEADING, 0, 0));
51         JPanel panel = new JPanel(new BorderLayout());
52         panel.add(getTypeGraphics(type));
53         add(panel);
54     }
55
56     private JPanel getTypeGraphics(lang.type.Type type) {
57         switch (type.getTypeCode()) {
58             case BOOL:
59                 return getBoolGraphics((TypeBool)type);
60             case AUTO:
61                 return getAutoGraphics((TypeAuto)type);
62             case LIST:
63                 return getListGraphics((TypeList)type);
64             case NUMBER:
65                 return getNumberGraphics((TypeNumber)type);
66             case RECORD:
67                 return getRecordGraphics((TypeRecord)type);
68             case STRING:
69                 return getStringGraphics((.TypeString)type);
70             case TIME:
71                 return getTimeGraphics((TypeTime)type);
72             case ERROR:
73                 return getErrorGraphics((TypeError)type);
```

```
74     case UNKNOWN:
75         return getUnknownGraphics((TypeUnknown)type);
76     default:
77         throw new RuntimeException("unexpected type code");
78     }
79 }
80
81 private class PrimitiveTextArea extends JTextArea {
82     PrimitiveTextArea(int w, int h) {
83         super(w,h);
84     }
85 }
86
87 private JTextArea getTextArea(int x, int y, String text, lang.type.Type type) {
88     JTextArea area;
89     switch (type.getTypeCode()) {
90         case AUTO:
91         case LIST:
92         case RECORD:
93             area = new JTextArea(x,y);
94             break;
95         case TIME:
96         case STRING:
97         case NUMBER:
98         case BOOL:
99         case UNKNOWN:
100        case ERROR:
101            area = new PrimitiveTextArea(x,y);
102            break;
103        default:
104            throw new RuntimeException("invalid type code");
105        }
106
107     area.setFont(UtilFont.textField);
108     area.setEditable(false);
109     area.setText(text);
110     DefaultCaret caret = (DefaultCaret)area.getCaret();
111     caret.setUpdatePolicy(DefaultCaret.ALWAYS_UPDATE);
112     area.setBorder(BorderFactory.createLoweredBevelBorder());
113     return area;
114 }
115
116 private JPanel getRecordGraphics(TypeRecord type) {
117     JPanel ret = new JPanel();
118
119     GridBagLayout columnLayout = new GridBagLayout();
120     GridBagConstraints constraints = new GridBagConstraints();
121     ret.setLayout(columnLayout);
122
123     List<RecordColumn> cols = type.getColumnsInOrder();
124     int xIdx = 0;
125     for (RecordColumn col : cols) {
126         JTextArea text = getTextArea(1,6,col.getId(),type);
127         text.setBackground(new Color(0xe0, 0xff, 0xff));
128         JPanel child = getTypeGraphics(col.getType());
```

```
129
130     constraints.fill = GridBagConstraints.BOTH;
131     constraints.weightx = 0;
132     constraints.weighty = 0;
133     constraints.gridx = xIdx;
134     constraints.gridy = 0;
135     constraints.gridwidth = 1;
136     constraints.gridheight = 1;
137     JPanel dummy = new JPanel(new BorderLayout());
138     columnLayout.setConstraints(dummy, constraints);
139     dummy.add(text, BorderLayout.CENTER);
140     ret.add(dummy);
141
142     constraints.fill = GridBagConstraints.BOTH;
143     constraints.weightx = 100;
144     constraints.weighty = 100;
145     constraints.gridx = xIdx;
146     constraints.gridy = 1;
147     constraints.gridwidth = 1;
148     constraints.gridheight = 1;
149     dummy = new JPanel(new BorderLayout());
150     columnLayout.setConstraints(dummy, constraints);
151     dummy.add(child, BorderLayout.CENTER);
152     ret.add(dummy);
153
154     xIdx++;
155 }
156
157     return ret;
158 }
159
160     private JPanel getAutoGraphics(TypeAuto type) {
161         JPanel ret = new JPanel(new BorderLayout());
162         JTextArea area = getTextArea(1,2,"Auto",type);
163         area.setBackground(new Color(0xFF, 0xCC, 0xCC));
164         ret.add(area, BorderLayout.LINE_START);
165         ret.add(getTypeGraphics(type.getType()), BorderLayout.LINE_END);
166         return ret;
167     }
168
169     private JPanel getListGraphics(TypeList type) {
170         JPanel ret = new JPanel(new BorderLayout());
171         JTextArea area = getTextArea(1,2,"List",type);
172         area.setBackground(new Color(0xCC, 0xFF, 0xCC));
173         ret.add(area, BorderLayout.LINE_START);
174         ret.add(getTypeGraphics(type.getChildType()), BorderLayout.LINE_END);
175         return ret;
176     }
177
178     private JPanel getLeafTypeGraphics(String name, lang.type.Type type) {
179         JPanel ret = new JPanel(new BorderLayout());
180         JTextArea area = getTextArea(1,5,name,type);
181         area.setBackground(new Color(0xd3, 0xd3, 0xd3));
182         ret.add(area);
183         return ret;
```

```

184     }
185
186     private JPanel getUnknownGraphics(TypeUnknown type) {
187         return getLeafTypeGraphics("Unknown",type);
188     }
189
190     private JPanel getErrorGraphics(TypeError type) {
191         return getLeafTypeGraphics("Error",type);
192     }
193
194     private JPanel getTimeGraphics(TypeTime type) {
195         return getLeafTypeGraphics("Time",type);
196     }
197
198     private JPanel getStringGraphics(TypeString type) {
199         return getLeafTypeGraphics("String",type);
200     }
201
202     private JPanel getBoolGraphics(TypeBool b) {
203         return getLeafTypeGraphics("Boolean",b);
204     }
205
206     private JPanel getNumberGraphics(TypeNumber n) {
207         return getLeafTypeGraphics("Number",n);
208     }
209
210     public static void showTypeDialog(Type type, Window parentWindow, boolean isOutput) {
211         JDialog dialog = new JDialog(parentWindow);
212         dialog.getContentPane().setLayout(new GridBagLayout());
213         dialog.setSize(500, 200);
214         dialog.setLocationRelativeTo(parentWindow);
215         ShowTypePanel panel = new ShowTypePanel(type, isOutput);
216         dialog.getContentPane().setBackground(Color.white);
217         dialog.getContentPane().add(panel);
218         dialog.setModal(true);
219         dialog.setVisible(true);
220     }
221 }
```

LISTING E.72: gui>ShowTypePanel.java

```

1 package gui.dataflow;
2
3 import java.util.ArrayList;
4 import java.util.Arrays;
5 import java.util.HashMap;
6 import java.util.List;
7 import java.util.Map;
8
9 import lang.type.RecordColumn;
10 import lang.type.Type;
11 import lang.type.TypeCode;
12 import lang.type.TypeRecord;
13
14 public class DataPartUtil {
```

```
15     public static TypeRecord getAugmentType(Type left, String leftLabel, Type right, String
16         rightLabel) {
17         if (left.getTypeCode() != TypeCode.RECORD) {
18             left = new TypeRecord(new ArrayList<>(Arrays.asList(new RecordColumn(leftLabel, left)))
19             ));
20         }
21     }
22
23     TypeRecord leftRec = (TypeRecord)left;
24     TypeRecord rightRec = (TypeRecord)right;
25
26     Map<String,Type> leftCols = leftRec.getColumns();
27     Map<String,Type> rightCols = rightRec.getColumns();
28     Map<String,String> leftRenames = new HashMap<>();
29     Map<String,String> rightRenames = new HashMap<>();
30     for (String key : leftCols.keySet()) {
31         if (rightCols.containsKey(key)) {
32             leftRenames.put(key, leftLabel+"_"+key);
33             rightRenames.put(key, rightLabel+"_"+key);
34         }
35     }
36
37     List<RecordColumn> finalColumns = new ArrayList<>();
38
39     List<RecordColumn> leftInOrder = leftRec.getColumnsInOrder();
40     for (int i = 0; i < leftInOrder.size(); i++) {
41         RecordColumn col = leftInOrder.get(i);
42         String name = leftRenames.get(col.getId());
43         if (name == null) {
44             finalColumns.add(col);
45         } else {
46             finalColumns.add(new RecordColumn(name, col.getType()));
47         }
48     }
49
50     List<RecordColumn> rightInOrder = rightRec.getColumnsInOrder();
51     for (int i = 0; i < rightInOrder.size(); i++) {
52         RecordColumn col = rightInOrder.get(i);
53         String name = rightRenames.get(col.getId());
54         if (name == null) {
55             finalColumns.add(col);
56         } else {
57             finalColumns.add(new RecordColumn(name, col.getType()));
58         }
59     }
60
61     return new TypeRecord(finalColumns);
62 }
63 }
```

LISTING E.73: gui/dataflow/DataPartUtil.java

```
1 package gui.dataflow;
2
3 @SuppressWarnings("serial")
4 public class InvalidDataPartException extends Exception {
5     public InvalidDataPartException(String msg) {
6         super(msg);
7     }
8 }
```

LISTING E.74: gui/dataflow/InvalidDataPartException.java

```
1 package gui.dataflow;
2
3 import gui.dataflow.DataPart.InputArrow;
4
5 import java.util.ArrayList;
6 import java.util.HashSet;
7 import java.util.Set;
8 import java.util.function.Consumer;
9
10 public class GraphUtil {
11     /* Returns true if there is no cycle in the graph. */
12     public static boolean depthTraversal(DataPart root, Consumer<DataPart> operation) {
13         DepthTraversal t = new DepthTraversal(operation);
14         return t.visit(root);
15     }
16
17     private static class DepthTraversal {
18         Set<DataPart> visited = new HashSet<>();
19         Consumer<DataPart> operation;
20
21         private DepthTraversal(Consumer<DataPart> operation) {
22             this.operation = operation;
23         }
24
25         private boolean visit(DataPart node) {
26             if (!visited.add(node)) {
27                 return false;
28             }
29             this.operation.accept(node);
30             boolean ret = true;
31             for (DataPart.OutputArrow a : node.outputArrows) {
32                 for (DataPart.InputArrow c : a.getConnections()) {
33                     ret = ret && visit(c.getDataPart());
34                 }
35             }
36             visited.remove(node);
37             return ret;
38         }
39     }
40
41     public enum ConnectResult {SUCCESS, CYCLE, MULTI_INPUT};
42     public static ConnectResult connectArrows(DataPart.InputArrow inputArrow, DataPart.OutputArrow
        outputArrow) {
```

```

43     if(inputArrow.connect(outputArrow)){
44         outputArrow.connect(inputArrow);
45         if (!GraphUtil.depthTraversal(outputArrow.getDataPart(), (n) -> {})) {
46             outputArrow.removeConnection(inputArrow);
47             inputArrow.removeConnection();
48             return ConnectResult.CYCLE;
49         }
50         return ConnectResult.SUCCESS;
51     }
52     return ConnectResult.MULTI_INPUT;
53 }
54
55 public static void disconnectArrow(DataPart.InputArrow inputArrow, DataPart.OutputArrow
56 outputArrow){
57     if(!inputArrow.connect(outputArrow)){
58         outputArrow.removeConnection(inputArrow);
59         inputArrow.removeConnection();
60         inputArrow.setTypeConnection(null);
61     }
62 }
63 public static void disconnectAllArrows(DataPart dataPart){
64     ArrayList<DataPart.OutputArrow> outputArrows = dataPart.outputArrows;
65     for(DataPart.OutputArrow arrowOut: outputArrows){
66         ArrayList<InputArrow> connectInputArrows = arrowOut.getConnections();
67         InputArrow[] inputArrowArray = connectInputArrows.toArray(new InputArrow[
68             connectInputArrows.size()]);
69         for(DataPart.InputArrow arrowIn: inputArrowArray){
70             disconnectArrow(arrowIn, arrowOut);
71         }
72     }
73     ArrayList<DataPart.InputArrow> inputArrows = dataPart.inputArrows;
74     for(DataPart.InputArrow arrowIn: inputArrows){
75         DataPart.OutputArrow arrowOut = arrowIn.getConnection();
76         if (arrowOut != null){
77             disconnectArrow(arrowIn, arrowOut);
78         }
79     }
}

```

LISTING E.75: gui/dataflow/GraphUtil.java

```

1 package gui.dataflow;
2
3 import gui.DataFlowPanel;
4 import gui.MainWindow;
5
6 import java.awt.BasicStroke;
7 import java.awt.Color;
8 import java.awt.Font;
9 import java.awt.Graphics;
10 import java.awt.Graphics2D;
11 import java.awt.Point;
12 import java.awt.Rectangle;
13 import java.awt.Stroke;

```

```
14 import java.awt.Window;
15 import java.awt.font.FontRenderContext;
16 import java.awt.font.TextLayout;
17 import java.awt.geom.AffineTransform;
18 import java.awt.geom.Line2D;
19 import java.awt.geom.Rectangle2D;
20 import java.awt.geom.RoundRectangle2D;
21 import java.util.ArrayList;
22 import java.util.Arrays;
23 import java.util.List;
24
25 import util.ColorTheme;
26 import util.UtilFont;
27 import lang.type.Type;
28 import lang.type.TypeAutoInputListener;
29 import lang.type.TypeAutoOutputListener;
30 import lang.type.TypeConnection;
31 import lang.type.TypeUnknown;
32
33 public abstract class DataPart {
34     private final static FontRenderContext renderContext =
35         new FontRenderContext(new AffineTransform(), true, false);
36     private final int diameter = 10;
37     private Point pointBox;
38     private static final Font arrowFont = UtilFont.textFont;
39     private static final Font titleFont = UtilFont.titleFont;
40     private static final Font subTitleFont = UtilFont.smallItalicFont;
41     private static final int minLineHeight = 10;
42     private static final int maxLineHeight = 80;
43     private static final int pointBoxTitleRoom = 4;
44     private static final int pointBoxTitleHeightRoom = 10;
45     private static final int pointBoxTitleSubTitleSpace = 4;
46     private static final int arrowTextRoom = 7;
47     private static final int maxBoxWidth = 250;
48     public static final int minBoxWidth = 50;
49
50     private DataFlowPanel dataFlowPanel;
51
52     private int inputLineHight = 40;
53     private int outputLineHight = 80;
54
55     public int boxHeight = 0;
56     private int boxWidth = 0;
57     /*internal*/ ArrayList<InputArrow> inputArrows;
58     /*internal*/ ArrayList<OutputArrow> outputArrows;
59
60     private ArrayList<Port> inputPorts;
61     private ArrayList<Port> outputPorts;
62
63     private Rectangle2D border = new Rectangle2D.Double();
64     private boolean drawHighlightBorder = false;
65     private boolean drawFocusBorder = false;
66
67     private String shortStringTitle;
68     private String stringTitle;
```

```
69     private TextLayout subTitle;
70     private TextLayout titleLayout;
71     private TextLayout shortTitleLayout;
72     private float periodLayoutWidth;
73
74     private int sequenceNumber = -1;
75
76     public DataPart(Port[] im, Port[] om, String dpTitle, String subTitle, DataFlowPanel dfp,
77                     Graphics g) {
78         dataFlowPanel = dfp;
79         pointBox = new Point(0, 0);
80         initialize(Port.clone(im), Port.clone(om), dpTitle, subTitle);
81     }
82
83     private void initialize(Port[] im, Port[] om, String dpTitle, String subTitle) {
84         inputPorts = new ArrayList<>(Arrays.asList(im));
85         outputPorts = new ArrayList<>(Arrays.asList(om));
86
87         inputArrows = new ArrayList<>();
88         outputArrows = new ArrayList<>();
89
90         inputLineHeight = getLineHeightSetLabels(inputPorts);
91         outputLineHeight = getLineHeightSetLabels(outputPorts);
92
93         // Initialize title and box width.
94         initializeWidth();
95         initializeTitle(dpTitle, subTitle);
96         initializeArrows();
97     }
98
99     public String getShortTitle() {
100        return shortStringTitle;
101    }
102
103    public String getTitle() {
104        return stringTitle;
105    }
106
107    public void setSequenceNumber(int n) {
108        this.sequenceNumber = n;
109    }
110
111    public int getSequenceNumber() {
112        return sequenceNumber;
113    }
114
115    public DataFlowPanel getDataFlowPanel() {
116        return dataFlowPanel;
117    }
118
119    private void initializeWidth() {
120        int inputs = inputPorts.size();
121        int outputs = outputPorts.size();
122        boxWidth = Math.max((int) (25 * 1.25 * Math.max(inputs, outputs)), minBoxWidth);
123    }
```

```
123
124     public void disconnectInputTypeIfConnected(String name) {
125         int index = -1;
126         for (int i = 0; i < inputPorts.size(); i++) {
127             if (inputPorts.get(i).getName().equals(name)) {
128                 index = i;
129                 break;
130             }
131         }
132         if (index == -1)
133             return;
134         OutputArrow a = inputArrows.get(index).getConnection();
135         GraphUtil.disconnectArrow(inputArrows.get(index), a);
136     }
137
138     protected void disconnectAllInputPortsIfConnected() {
139         for (Port p : inputPorts)
140             disconnectInputTypeIfConnected(p.getName());
141     }
142
143     public void disconnectOutputTypeIfConnected(String name) {
144         int index = -1;
145         for (int i = 0; i < outputPorts.size(); i++) {
146             if (outputPorts.get(i).getName().equals(name)) {
147                 index = i;
148                 break;
149             }
150         }
151         if (index == -1)
152             return;
153
154         OutputArrow arr = outputArrows.get(index);
155         ArrayList<InputArrow> inputConns = new ArrayList<>(arr.getConnections());
156         for (InputArrow conn : inputConns) {
157             GraphUtil.disconnectArrow(conn, arr);
158         }
159     }
160
161     private void initializeSubTitle(String subTitle) {
162         if (subTitle == null)
163             return;
164         if (subTitle.length() == 0)
165             subTitle = " ";
166         this.subTitle = new TextLayout(subTitle, subTitleFont, renderContext);
167     }
168
169     private void initializeTitle(String newTitle, String subTitle) {
170         initializeSubTitle(subTitle);
171
172         String text = newTitle;
173         if (text.length() == 0)
174             text = " ";
175
176         shortStringTitle = stringTitle = newTitle;
177         titleLayout = new TextLayout("." + text + ".", titleFont, renderContext);
```

```
178
179     text = text.substring(0, text.length()-1);
180     while (titleLayout.getBounds().getWidth() > maxBoxWidth - pointBoxTitleRoom/2) {
181         text = text.substring(0, text.length()-1);
182         shortStringTitle = text+"...";
183         titleLayout = new TextLayout("."+shortStringTitle+".", titleFont, renderContext);
184     }
185
186     boxWidth = Math.max((int)boxWidth, getTitleWidth());
187     boxHeight = pointBoxTitleSubTitleSpace + pointBoxTitleHeightRoom*2 +
188                 (int)titleLayout.getBounds().getHeight() + (int)this.subTitle.getBounds().
189                 getHeight();
190
191     shortTitleLayout = new TextLayout(shortStringTitle.equals("") ? " " : shortStringTitle,
192                                     titleFont, renderContext);
193     periodLayoutWidth = (float)new TextLayout(".", titleFont, renderContext).getBounds().
194     getWidth();
195
196 }
197
198 public int getTitleWidth() {
199     return (int)Math.max(titleLayout.getBounds().getWidth(), subTitle.getBounds().getWidth())+
200     pointBoxTitleRoom;
201 }
202
203
204 protected void addInputPort(Port port) {
205     addInputPort(port, null);
206 }
207
208 protected void addInputPort(Port port, TypeAutoInputListener listener) {
209     port = port.clone();
210     inputPorts.add(port);
211     inputLineHight = getLineHeightSetLabels(inputPorts);
212     initializeWidth();
213     boxWidth = Math.max(getTitleWidth(), boxWidth);
214     InputArrow arr = new InputArrow(this, 0, 0, diameter, inputLineHight, port.getType(),
215     listener);
216     arr.setText(port.label);
217     inputArrows.add(arr);
218     for (InputArrow a : inputArrows) {
219         a.setLineHeight(inputLineHight);
220     }
221     repositionArrows();
222 }
223
224 protected void addOutputPort(Port port) {
225     port = port.clone();
226     outputPorts.add(port);
227     outputLineHight = getLineHeightSetLabels(outputPorts);
228     initializeWidth();
```

```
228     boxWidth = Math.max(getTitleWidth(), boxWidth);
229     OutputArrow arr = new OutputArrow(this, 0, 0, diameter, outputLineHight, port.getType());
230     arr.setText(port.label);
231     outputArrows.add(arr);
232     for (OutputArrow a : outputArrows) {
233         a.setLineHeight(outputLineHight);
234     }
235     repositionArrows();
236 }
237
238 public void changeInputPortType(String oldName, Type newType) {
239     int index = inputPorts.indexOf(new Port(oldName, new TypeUnknown(""))));
240     Port prev = inputPorts.get(index);
241     changeInputPort(new Port(oldName, prev.getType()), new Port(oldName, newType));
242 }
243
244 public void changeInputPortName(String oldName, String newName) {
245     int index = inputPorts.indexOf(new Port(oldName, new TypeUnknown(""))));
246     Port prev = inputPorts.get(index);
247     changeInputPort(new Port(oldName, prev.getType()), new Port(newName, prev.getType()));
248 }
249
250 public void changeOutputPortType(String oldName, Type newType) {
251     int index = outputPorts.indexOf(new Port(oldName, new TypeUnknown(""))));
252     Port prev = outputPorts.get(index);
253     changeOutputPort(new Port(oldName, prev.getType()), new Port(oldName, newType));
254 }
255
256 public void changeOutputPortName(String oldName, String newName) {
257     int index = outputPorts.indexOf(new Port(oldName, new TypeUnknown(""))));
258     Port prev = outputPorts.get(index);
259     changeOutputPort(new Port(oldName, prev.getType()), new Port(newName, prev.getType()));
260 }
261
262 protected void changeOutputPort(Port oldPort, Port newPort) {
263     int index = outputPorts.indexOf(oldPort);
264     outputPorts.get(index).label = newPort.getName();
265     outputPorts.get(index).setName(newPort.getName());
266     outputPorts.get(index).setType(newPort.getType());
267     outputLineHight = getLineHeightSetLabels(outputPorts);
268     initializeWidth();
269     boxWidth = Math.max(getTitleWidth(), boxWidth);
270     outputArrows.get(index).setText(outputPorts.get(index).label);
271     outputArrows.get(index).setType(outputPorts.get(index).getType());
272     for (OutputArrow a : outputArrows) {
273         a.setLineHeight(outputLineHight);
274     }
275     repositionArrows();
276 }
277
278 protected void changeInputPort(Port oldPort, Port newPort) {
279     int index = inputPorts.indexOf(oldPort);
280     inputPorts.get(index).label = newPort.getName();
281     inputPorts.get(index).setName(newPort.getName());
282     inputPorts.get(index).setType(newPort.getType());
```

```
283     inputLineHeight = getLineHeightSetLabels(inputPorts);
284     initializeWidth();
285     boxWidth = Math.max(getTitleWidth(), boxWidth);
286     inputArrows.get(index).setText(inputPorts.get(index).label);
287     inputArrows.get(index).setType(inputPorts.get(index).getType());
288     for (InputArrow a : inputArrows) {
289         a.setLineHeight(inputLineHeight);
290     }
291     repositionArrows();
292 }
293
294 protected void removeOutputPort(Port port) {
295     int index = outputPorts.indexOf(port);
296
297     OutputArrow arr = outputArrows.get(index);
298     ArrayList<InputArrow> inputConns = new ArrayList<>(arr.getConnections());
299     for (InputArrow conn : inputConns) {
300         GraphUtil.disconnectArrow(conn, arr);
301     }
302
303     outputPorts.remove(index);
304     outputArrows.remove(index);
305     outputLineHeight = getLineHeightSetLabels(outputPorts);
306     initializeWidth();
307     boxWidth = Math.max(getTitleWidth(), boxWidth);
308     for (OutputArrow a : outputArrows) {
309         a.setLineHeight(outputLineHeight);
310     }
311     repositionArrows();
312 }
313
314 protected void removeInputPort(Port port) {
315     int index = inputPorts.indexOf(port);
316
317     InputArrow arr = inputArrows.get(index);
318     OutputArrow conn = arr.getConnection();
319     if (conn != null)
320         GraphUtil.disconnectArrow(arr, conn);
321
322     inputPorts.remove(index);
323     inputArrows.remove(index);
324     inputLineHeight = getLineHeightSetLabels(inputPorts);
325     initializeWidth();
326     boxWidth = Math.max(getTitleWidth(), boxWidth);
327     for (InputArrow a : inputArrows) {
328         a.setLineHeight(inputLineHeight);
329     }
330     repositionArrows();
331 }
332
333 protected void removeAllInputPorts() {
334     ArrayList<Port> ports = new ArrayList<>(inputPorts);
335     for (Port p : ports) {
336         removeInputPort(p);
337     }

```

```
338     }
339
340     protected void addAllInputPorts(Port[] ports) {
341         for (Port p : ports)
342             addInputPort(p);
343     }
344
345     protected boolean isOutputPortConnected(Port p) {
346         int index = outputPorts.indexOf(p);
347         OutputArrow a = outputArrows.get(index);
348         return a.getConnections().size() > 0;
349     }
350
351     protected boolean isInputPortConnected(Port p) {
352         int index = inputPorts.indexOf(p);
353         InputArrow a = inputArrows.get(index);
354         return a.getConnection() != null;
355     }
356
357     public String getOutputName(DataPart.OutputArrow a) {
358         int index = outputArrows.indexOf(a);
359         return outputPorts.get(index).getName();
360     }
361
362     public String getInputName(DataPart.InputArrow a) {
363         int index = inputArrows.indexOf(a);
364         return inputPorts.get(index).getName();
365     }
366
367     private int getLineHeightSetLabels(ArrayList<Port> a) {
368         double height = minLineHeight;
369         for (int i = 0; i < a.size(); i++) {
370             if (a.get(i).label.length() == 0 || a.get(i).label.startsWith("!"))
371                 a.get(i).label = " ";
372             String s = a.get(i).label;
373             TextLayout layout = new TextLayout(s, arrowFont, renderContext);
374             s = s.substring(0, s.length()-1);
375             while (layout.getBounds().getWidth() > maxLineHeight-arrowTextRoom) {
376                 a.get(i).label = s+"...";
377                 s = s.substring(0, s.length()-1);
378                 layout = new TextLayout(a.get(i).label, arrowFont, renderContext);
379             }
380             height = Math.max(layout.getBounds().getWidth()-arrowTextRoom, height);
381         }
382         return (int)height;
383     }
384
385     public Point getPointBoxLocation() {
386         return new Point(pointBox.getLocation());
387     }
388
389     public void setPointBoxLocation(Point p) {
390         Point current = getPointBoxLocation();
391         current.x = -current.x;
392         current.y = -current.y;
```

```
393     movePointBox(current);
394     movePointBox(p);
395 }
396
397 public void movePointBox(Point p){
398     int xdelta = pointBox.x+p.x <= 0 ? -pointBox.x : p.x;
399     int ydelta = pointBox.y+p.y <= 0 ? -pointBox.y : p.y;
400     pointBox.x += p.x;
401     pointBox.y += p.y;
402     if(pointBox.x <=0){
403         pointBox.x = 0;
404     }
405     if(pointBox.y <=0){
406         pointBox.y = 0;
407     }
408     for (InputArrow a : inputArrows) {
409         a.setX(a.getX()+xdelta);
410         a.setY(a.getY()+ydelta);
411     }
412     for (OutputArrow a : outputArrows) {
413         a.setX(a.getX()+xdelta);
414         a.setY(a.getY()+ydelta);
415     }
416 }
417
418 public boolean isMouseInBound(Point p){
419     int mouseXPos = p.x;
420     int mouseYPos = p.y;
421     if(border.contains(mouseXPos, mouseYPos)){
422         return true;
423     }
424     return false;
425 }
426
427 private void setLowerRight(Point lowerRight, Arrow a) {
428     Point oval = a.getOvalCenter();
429     int xx = oval.x+diameter/2;
430     int yy = oval.y+diameter/2;
431
432     if (xx > lowerRight.x)
433         lowerRight.x = xx;
434     if (yy > lowerRight.y)
435         lowerRight.y = yy;
436 }
437
438 private void setUpperLeft(Point upperLeft, Arrow a) {
439     Point oval = a.getOvalCenter();
440     int xx = oval.x-diameter/2;
441     int yy = oval.y-diameter/2;
442     if (xx < upperLeft.x)
443         upperLeft.x = xx;
444     if (yy < upperLeft.y)
445         upperLeft.y = yy;
446 }
447
```

```
448     public Rectangle getBoundingRectangleWithLines() {
449         double x = pointBox.getX()-diameter-5;
450         double y = pointBox.getY()-diameter*2-inputLineHight-5;
451         double w = boxWidth+diameter*2+10;
452         double h = boxHeight+inputLineHight+outputLineHight+diameter*4+10;
453         Point upperLeft = new Point((int)x-1,(int)y-1);
454         Point lowerRight = new Point((int) (x+w) + 1, (int) (y+h) + 1);
455         for (InputArrow i : inputArrows) {
456             OutputArrow o = i.outputArrow;
457             if (o != null) {
458                 setLowerRight(lowerRight, o);
459                 setUpperLeft(upperLeft, o);
460             }
461         }
462         for (OutputArrow o : outputArrows) {
463             for (InputArrow i : o.getConnections()) {
464                 setLowerRight(lowerRight, i);
465                 setUpperLeft(upperLeft, i);
466             }
467         }
468         x = upperLeft.x-6;
469         y = upperLeft.y-6;
470         w = lowerRight.x-upperLeft.x + 12;
471         h = lowerRight.y-upperLeft.y + 12;
472         return new Rectangle((int)x,(int)y,(int)w,(int)h);
473     }
474
475     private Rectangle2D getBoundingRectangle() {
476         double x = pointBox.getX()-diameter;
477         double y = pointBox.getY()-diameter*2-inputLineHight;
478         double w = boxWidth+diameter*2;
479         double h = boxHeight+inputLineHight+outputLineHight+diameter*4;
480         return new Rectangle2D.Double(x, y, w, h);
481     }
482
483     private void drawRoundRect(Graphics2D g2, double x, double y, double w, double h, Color background) {
484         Color borderColor = Color.black;
485         RoundRectangle2D bound = new RoundRectangle2D.Double(x,y,w,h,10,10);
486         g2.setColor(background);
487         g2.fill(bound);
488         g2.setColor(borderColor);
489         g2.draw(bound);
490     }
491
492     private void drawBoundBox(Graphics2D g2) {
493         Color color;
494         int is = inputPorts.size();
495         int os = outputPorts.size();
496         double x = pointBox.getX()-diameter;
497         double y = pointBox.getY()-diameter*2-inputLineHight;
498         double w = boxWidth+diameter*2;
499         double h = boxHeight+inputLineHight+outputLineHight+diameter*4;
500
501         if (is > 0 && os > 0) {
```

```
502         color = ColorTheme.TRANS_YELLOW;
503     } else if (is > 0) {
504         color = ColorTheme.TRANS_BLUE;
505         h = boxHeight+inputLineHeight+diameter*3;
506     } else if (os > 0) {
507         color = ColorTheme.TRANS_GREEN;
508         y = pointBox.getY()-diameter;
509         h = boxHeight+outputLineHeight+diameter*3;
510     } else {
511         color = ColorTheme.TRANS_RED;
512         y = pointBox.getY()-diameter;
513         h = boxHeight+diameter*2;
514     }
515     drawRoundRect(g2,x,y,w,h,color);
516     border = new Rectangle2D.Double(x, y, w, h);
517
518     if(drawFocusBorder){
519         RoundRectangle2D highlight = new RoundRectangle2D.Double(x-1,y-1,w+2,h+2,10,10);
520         Stroke d = g2.getStroke();
521         g2.setStroke(new BasicStroke(4));
522         //g2.setColor(new Color(0x66, 0x66, 0, 127));
523         g2.setColor(ColorTheme.HOVERED);
524         g2.draw(highlight);
525         g2.setStroke(d);
526     }
527     if (drawHighlightBorder) {
528         RoundRectangle2D highlight = new RoundRectangle2D.Double(x-1,y-1,w+2,h+2,10,10);
529         Stroke d = g2.getStroke();
530         g2.setStroke(new BasicStroke(3));
531         //g2.setColor(new Color(0xff, 0, 0, 127));
532         g2.setColor(ColorTheme.SELECTED);
533         g2.draw(highlight);
534         g2.setStroke(d);
535     }
536 }
537
538     public boolean isHighlighted() {
539         return drawHighlightBorder;
540     }
541     public void setDrawFocusBorder(boolean b){
542         drawFocusBorder = b;
543     }
544     public void setDrawHighlightBorder(boolean b) {
545         drawHighlightBorder = b;
546     }
547
548     public void paint(Graphics g){
549         Graphics2D g2 = (Graphics2D)g;
550
551         //draw border box
552         g2.setColor(Color.BLACK);
553         drawBoundBox(g2);
554         //draw box
555         drawRoundRect(g2, pointBox.getX(), pointBox.getY(), boxWidth, boxHeight, new Color(0xFF, 0
556         xFF, 0xFF, 127));
```

```
556     //draw title
557
558     Rectangle2D subTitleBounds = subTitle.getBounds();
559     Rectangle2D titleBounds = titleLayout.getBounds();
560
561     int titleHeight = (int)Math.ceil(subTitleBounds.getHeight() +
562             titleBounds.getHeight() + pointBoxTitleSubTitleSpace);
563     float startY = (float)pointBox.getY() + boxHeight/2.0f - titleHeight/2.0f + (float)
564     titleBounds.getHeight();
565
566     shortTitleLayout.draw(g2, 2*periodLayoutWidth + (float)pointBox.getX() + boxWidth/2 - (int
567     ) (titleBounds.getWidth()/2), startY);
568     float yy = startY + (float)subTitleBounds.getHeight() + (float)pointBoxTitleSubTitleSpace;
569     subTitle.draw(g2, (float)pointBox.getX() + boxWidth/2 - (int) (subTitleBounds.getWidth()
570     /2), yy);
571
572     //draw input and output arrows
573     for (InputArrow a : inputArrows)
574         a.paintArrow(g2);
575     for (OutputArrow a : outputArrows)
576         a.paintArrow(g2);
577 }
578
579     public void paintOutputLines(Graphics g) {
580         Graphics2D g2 = (Graphics2D)g;
581         Stroke orig = g2.getStroke();
582         g2.setStroke(new BasicStroke(2));
583         for (OutputArrow o : outputArrows) {
584             for (InputArrow i : o.getConnections()) {
585                 Line2D line = new Line2D.Double(o.getOvalCenter(), i.getOvalCenter());
586                 g2.draw(line);
587             }
588         }
589         g2.setStroke(orig);
590     }
591
592     public ArrayList<LineConnectPart> getOutputLines() {
593         ArrayList<LineConnectPart> outputLines = new ArrayList<LineConnectPart>();
594         for (OutputArrow o : outputArrows) {
595             for (InputArrow i : o.getConnections()) {
596                 outputLines.add(new LineConnectPart(i, o));
597             }
598         }
599         return outputLines;
600     }
601
602     private void initializeArrows() {
603         int inputs = inputPorts.size();
604         int outputs = outputPorts.size();
605         //initialize input arrows
606         if (inputs > 0)
607             inputArrows.add(new InputArrow(this, 0, 0, diameter, inputLineHight, inputPorts.get(0)
608             .getType(), null));
609         for(int i=0; i<inputs; i++){
610             inputArrows.get(i).setText(inputPorts.get(i).label);
```

```
607         if (i < inputs-1) {
608             inputArrows.add(new InputArrow(this, 0, 0, diameter, inputLineHight, inputPorts.
609             get(i+1).getType(), null));
610         }
611     }
612     //initialize output arrows
613     if (outputs > 0)
614         outputArrows.add(new OutputArrow(this, 0, 0, diameter, outputLineHight, outputPorts.
615         get(0).getType()));
616         for(int j=0; j<outputs; j++){
617             outputArrows.get(j).setText(outputPorts.get(j).label);
618             if (j < outputs-1) {
619                 outputArrows.add(new OutputArrow(this, 0, 0, diameter, outputLineHight,
620                 outputPorts.get(j+1).getType()));
621             }
622         }
623     repositionArrows();
624 }
625
626 private void repositionArrows() {
627     int inputs = inputPorts.size();
628     int outputs = outputPorts.size();
629     int divideInput = (int) Math.round((double) boxWidth / inputs);
630     int divideoutput = (int) Math.round((double) boxWidth / outputs);
631     //initialize input arrows
632     if (inputs > 0) {
633         inputArrows.get(0).setX((int)pointBox.getX()+divideInput/2-diameter/2);
634         inputArrows.get(0).setY((int)pointBox.getY()-diameter*3-inputLineHight+diameter/2);
635     }
636     for(int i=0; i<inputs; i++){
637         int xPosIn = inputArrows.get(i).getX();
638         if (i < inputs-1) {
639             inputArrows.get(i+1).setX(xPosIn+divideInput);
640             inputArrows.get(i+1).setY((int)pointBox.getY()-diameter*3-inputLineHight+diameter
641             /2);
642         }
643     }
644     //initialize output arrows
645     if (outputs > 0) {
646         outputArrows.get(0).setX((int)pointBox.getX()+divideoutput/2-diameter/2);
647         outputArrows.get(0).setY((int)pointBox.getY()+boxHeight-diameter/2);
648     }
649     for(int j=0; j<outputs; j++){
650         int xPosOut = outputArrows.get(j).getX();
651         if (j < outputs-1) {
652             outputArrows.get(j+1).setX(xPosOut+divideoutput);
653             outputArrows.get(j+1).setY((int)pointBox.getY()+boxHeight-diameter/2);
654         }
655     }
656 }
657
658 public OutputArrow getOutputArrow(Point p) {
659     for (OutputArrow a : outputArrows) {
660         if (a.isInsideArrowOval(p))
661             return a;
```

```
658     }
659     return null;
660 }
661 public OutputArrow getOutputArrow(String name) {
662     int index = outputPorts.indexOf(new Port(name, new TypeUnknown("")));
663     return outputArrows.get(index);
664 }
665 public InputArrow getInputArrow(Point p) {
666     for (InputArrow a : inputArrows) {
667         if (a.isInsideArrowOval(p))
668             return a;
669     }
670     return null;
671 }
672 public InputArrow getInputArrow(String name) {
673     int index = inputPorts.indexOf(new Port(name, new TypeUnknown("")));
674     return inputArrows.get(index);
675 }
676 public ArrayList<OutputArrow> getOutputArrows() {
677     return new ArrayList<>(outputArrows);
678 }
679 public ArrayList<InputArrow> getInputArrows() {
680     return new ArrayList<>(inputArrows);
681 }
682
683 public void setInputArrowListeners(List<TypeAutoInputListener> ls) {
684     if (ls.size() != inputArrows.size())
685         throw new IllegalArgumentException("invalid number of listeners "+ls.size()+" != "+inputArrows.size());
686     for (int i = 0; i < ls.size(); i++) {
687         inputArrows.get(i).autoListener = ls.get(i);
688     }
689 }
690
691 public void setInputArrowListener(String name, TypeAutoInputListener listener) {
692     int index = inputPorts.indexOf(new Port(name, new TypeUnknown("")));
693     inputArrows.get(index).autoListener = listener;
694 }
695
696 public void setOutputArrowListener(String name, TypeAutoOutputListener listener) {
697     int index = outputPorts.indexOf(new Port(name, new TypeUnknown("")));
698     outputArrows.get(index).autoListener = listener;
699 }
700
701 public abstract static class Arrow {
702     DataPart dataPart;
703     int xPos = 0;
704     int yPos = 0;
705     int diameter = 0;
706     int lineHeight = 0;
707     Point ovalCenter = new Point(0,0);
708     String text;
709     Type type;
710
711     final int MAX_TEXT_LENGTH = 8;
```

```
712
713     Arrow(DataPart dp, int xPos, int yPos, int diameter, int lineHeight, Point ovalCenter,
714     Type type) {
715         dataPart = dp;
716         this.ovalCenter = new Point(ovalCenter.x, ovalCenter.y);
717         this.xPos = xPos;
718         this.yPos = yPos;
719         this.diameter = diameter;
720         this.lineHeight = lineHeight;
721         this.type = type;
722     }
723     public DataPart getDataPart() {
724         return dataPart;
725     }
726     void setText(String t){
727         this.text = t;
728     }
729     String getText(){
730         return this.text;
731     }
732     void setX(int xPos){
733         int diff = xPos - this.xPos;
734         this.xPos = xPos;
735         ovalCenter.x += diff;
736     }
737     int getX(){
738         return xPos;
739     }
740     int getY() {
741         return yPos;
742     }
743     void setY(int yPos){
744         int diff = yPos - this.yPos;
745         this.yPos = yPos;
746         ovalCenter.y += diff;
747     }
748     void setDiameter(int diameter){
749         this.diameter = diameter;
750     }
751     abstract void setLineHeight(int lineHeight);
752
753     String getVisibleText() {
754         return this.text;
755     }
756     boolean isInsideArrowOval(Point p){
757         double dist = Math.sqrt(Math.pow(p.x-ovalCenter.x, 2) + Math.pow(p.y-ovalCenter.y, 2))
758         ;
759         return dist <= ((diameter+4) / 2.0);
760     }
761     public Point getOvalCenter(){
762         return ovalCenter;
763     }
764     public Rectangle getBoundingOvalRect() {
765         return new Rectangle(ovalCenter.x-diameter/2-5,ovalCenter.y-diameter/2-5, diameter+10,
766         diameter+10);
```

```
764     }
765     public int getDiameter() {
766         return diameter;
767     }
768     public Type getType() {
769         return type;
770     }
771     public void setType(Type type) {
772         this.type = type;
773     }
774 }
775
776     public static class InputArrow extends Arrow{
777         TypeConnection typeConnection;
778         private OutputArrow outputArrow;
779         private TypeAutoInputListener autoListener;
780
781         public InputArrow(DataPart dp, int xPos, int yPos, int diameter, int lineHeight, Type type
782 , TypeAutoInputListener listener) {
783             super(dp, xPos, yPos, diameter, lineHeight, new Point(xPos+diameter/2, yPos+diameter
784 /2), type);
785             autoListener = listener;
786         }
787
788         public void removeConnection(){
789             if (autoListener != null)
790                 autoListener.autoTypeInputRemoved();
791             outputArrow = null;
792         }
793         public OutputArrow getConnection() {
794             return outputArrow;
795         }
796         public boolean connect(OutputArrow oa){
797             if(outputArrow == null){
798                 outputArrow = oa;
799                 return true;
800             }
801             return false;
802         }
803
804         public void paintArrow(Graphics2D g){
805             g.setColor(Color.BLACK);
806             g.fillOval(xPos, yPos, diameter, diameter);
807             g.setColor(Color.BLACK);
808             g.drawLine(xPos+diameter/2, yPos+diameter, xPos+diameter/2, yPos+diameter*2+lineHeight
809 );
810             g.setColor(Color.BLACK);
811             g.fillPolygon(new int[] {xPos, xPos+diameter, xPos+diameter/2}, new int[] {yPos+
812 diameter*2+lineHeight, yPos+diameter*2+lineHeight, yPos+diameter*3+lineHeight}, 3);
813
814             //draw text
815             AffineTransform defaultAt = g.getTransform();
816             g.rotate(Math.PI/2);
817             g.setColor(Color.BLACK);
818             g.setFont(DataPart.arrowFont);
```

```
815         //g.drawString(getVisibleText(), xPos+diameter, yPos+diameter);
816         g.drawString(getVisibleText(), yPos+diameter, -(xPos+diameter));
817         g.setTransform(defaultAt);
818     }
819
820     public void setTypeConnection(TypeConnection conn) {
821         typeConnection = conn;
822         if (conn != null) {
823             if (autoListener != null)
824                 conn.setTypeAutoListener(autoListener);
825             conn.autoConnect();
826         }
827     }
828
829     public TypeConnection getTypeConnection() {
830         return typeConnection;
831     }
832
833     @Override
834     void setLineHeight(int lineHeight) {
835         this.lineHeight = lineHeight;
836     }
837
838     public static class OutputArrow extends Arrow {
839
840         private ArrayList<InputArrow> inputArrows = new ArrayList<InputArrow>();
841         private TypeAutoOutputListener autoListener;
842
843
844         public OutputArrow(DataPart dp, int xPos, int yPos, int diameter, int lineHeight, Type
845             type) {
846             super(dp, xPos, yPos, diameter, lineHeight, new Point(xPos+diameter/2, yPos+diameter
847 *2+lineHeight+diameter/2), type);
848         }
849
850         public ArrayList<InputArrow> getConnections() {
851             return inputArrows;
852         }
853
854         public boolean removeConnection(InputArrow ia){
855             boolean ret = inputArrows.remove(ia);
856             if (autoListener != null && getType().hasAuto())
857                 autoListener.autoTypeOutputRemoved();
858             return ret;
859         }
860
861         public void connect(InputArrow ia){
862             inputArrows.add(ia);
863         }
864
865         public void paintArrow(Graphics2D g){
866             g.setColor(Color.BLACK);
867             g.fillPolygon(new int[] {xPos, xPos+diameter, xPos+diameter/2}, new int[] {yPos, yPos,
868             yPos+diameter}, 3);
869             g.setColor(Color.BLACK);
870             g.drawLine(xPos+diameter/2, yPos+diameter, xPos+diameter/2, yPos+diameter*2+lineHeight
871 );
872             g.setColor(Color.BLACK);
873             g.fillOval(xPos, yPos+diameter*2+lineHeight, diameter, diameter);
874             //draw text
875         }
876     }
877 }
```

```

866         AffineTransform defaultAt = g.getTransform();
867         g.rotate(Math.PI/2);
868         g.setColor(Color.BLACK);
869         g.setFont(DataPart.arrowFont);
870         //g.drawString("01234567.", xPos+diameter/2, yPos+diameter);
871         g.drawString(getVisibleText(), yPos+diameter, -(xPos+diameter));
872         g.setTransform(defaultAt);
873     }
874
875     @Override
876     void setLineHeight(int lineHeight) {
877         this.lineHeight = lineHeight;
878         ovalCenter = new Point(xPos+diameter/2, yPos+diameter*2+lineHeight+diameter/2);
879     }
880 }
881
882     public boolean isParameterTypeConnected(String param) {
883         return isInputPortConnected(new Port(param, new TypeUnknown(""))));
884     }
885
886     public boolean isResultTypeConnected(String result) {
887         return isOutputPortConnected(new Port(result, new TypeUnknown(""))));
888     }
889
890     public abstract boolean canFail();
891     public abstract void doubleClick(MainWindow mainWindow, Window parent);
892     public abstract boolean markDeletedIfAllowed();
893
894     public boolean isInsideOrIntersects(Rectangle r) {
895         Rectangle2D b = getBoundingRectangle();
896         return r.contains(b) || r.intersects(b);
897     }
898 }
```

LISTING E.76: gui/dataflow/DataPart.java

```

1 package gui.dataflow;
2
3 import gui.DataFlowPanel;
4
5 public interface IDataPartSubFlow {
6     DataFlowPanel getSubDataFlowPanel();
7 }
```

LISTING E.77: gui/dataflow/IDataPartSubFlow.java

```

1 package gui.dataflow;
2
3 import java.awt.Graphics;
4 import java.awt.Window;
5
6 import lang.type.TypeBool;
7 import lang.type.TypeNumber;
8 import gui.DataFlowPanel;
```

```
9 import gui.MainWindow;
10
11 public class DataPartArithmetic extends DataPart {
12     public static String subtitle = "Arithmetic";
13
14     public enum ArithmeticType {PLUS, MINUS, MUL, DIV, NEGATE, AND, OR, NOT};
15
16     private ArithmeticType arithmeticType;
17
18     private final static Port[] binaryBoolInputs =
19         {new Port("!0", new TypeBool()), new Port("!1", new TypeBool())};
20     private final static Port[] unaryBoolInputs =
21         {new Port("", new TypeBool())};
22     private final static Port[] boolOutputs = {new Port("", new TypeBool())};
23     private final static Port[] binaryInputs =
24         {new Port("!0", new TypeNumber()), new Port("!1", new TypeNumber())};
25     private final static Port[] unaryInputs =
26         {new Port("", new TypeNumber())};
27     private final static Port[] outputs = {new Port("", new TypeNumber())};
28
29     private static String getSubTitle(ArithmeticType type) {
30         switch (type) {
31             case AND:
32             case OR:
33             case NOT:
34                 return DataPartBoolean.subtitle;
35             default:
36                 return "Arithmetic";
37         }
38     }
39
40     private static Port[] getInputPorts(ArithmeticType type) {
41         switch (type) {
42             case AND:
43             case OR:
44                 return binaryBoolInputs;
45             case NOT:
46                 return unaryBoolInputs;
47             case NEGATE:
48                 return unaryInputs;
49             default:
50                 return binaryInputs;
51         }
52     }
53
54     private static String getArithmeticTitle(ArithmeticType type) {
55         switch (type) {
56             case PLUS:
57                 return "\u002B";
58             case MINUS:
59                 return "\u2212";
60             case MUL:
61                 return "\u00D7";
62             case DIV:
63                 return "\u00F7";
```

```
64     case NEGATE:
65         return "Negate";
66     case AND:
67         return "\u2227";
68     case OR:
69         return "\u2228";
70     case NOT:
71         return "\u00AC";
72     default:
73         throw new RuntimeException("unexpected arithmetic type " + type);
74     }
75 }
76
77 private static Port[] getOutputPorts(ArithmeticType type) {
78     switch (type) {
79     case AND:
80     case OR:
81     case NOT:
82         return boolOutputs;
83     default:
84         return outputs;
85     }
86 }
87
88
89 public DataPartArithmetic(ArithmeticType type, DataFlowPanel dfp, Graphics g) {
90     super(getInputPorts(type), getOutputPorts(type), getArithmeticTitle(type), getSubTitle(
91         type), dfp, g);
92     arithmeticType = type;
93 }
94
95 public ArithmeticType getType() {
96     return arithmeticType;
97 }
98
99 @Override
100 public boolean canFail() {
101     return getType() == ArithmeticType.DIV;
102 }
103
104 @Override
105 public void doubleClick(MainWindow mainWindow, Window parent) {}
106
107 @Override
108 public boolean markDeletedIfAllowed() {
109     return true;
110 }
```

LISTING E.78: gui/dataflow/DataPartArithmetic.java

```
1 package gui.dataflow;
2
3 import java.awt.FlowLayout;
4 import java.awt.Graphics;
```

```
5 import java.awt.Window;
6 import java.awt.event.WindowAdapter;
7 import java.awt.event.WindowEvent;
8
9 import javax.swing.JDialog;
10 import javax.swing.JOptionPane;
11 import javax.swing.JSplitPane;
12
13 import gui.DataFlowPanel;
14 import gui.MainWindow;
15 import gui.ShowTypePanel;
16 import gui.ValuePanel;
17 import lang.constant.Constant;
18 import lang.constant.InvalidConstantException;
19 import util.UtilScrollPane;
20
21 public class DataPartConstant extends DataPart {
22
23     private final static Port[] inputPorts = {};
24     @SuppressWarnings("unused")
25     private Port[] outputPorts;
26     private Constant constant;
27
28     public DataPartConstant(Constant value, DataFlowPanel dfp, Graphics g) {
29         this(value, new Port[]{new Port("", value.getType())}, dfp, g);
30     }
31
32     private DataPartConstant(Constant value, Port[] outputPorts, DataFlowPanel dfp, Graphics g) {
33         super(inputPorts, outputPorts, value.getShortDescription(), "Constant", dfp, g);
34         constant = value;
35         this.outputPorts = outputPorts;
36     }
37
38     public Constant getConstant() {
39         return constant;
40     }
41
42     @Override
43     public boolean canFail() {
44         return false;
45     }
46
47     @Override
48     public void doubleClick(MainWindow mainWindow, Window parent) {
49         JDialog dialog = new JDialog(parent);
50         dialog.setSize(500, 400);
51         dialog.setLocationRelativeTo(parent);
52         ShowTypePanel showTypePanel = new ShowTypePanel(constant.getType(), true);
53         showTypePanel.setLayout(new FlowLayout(FlowLayout.LEADING, 10, 10));
54         UtilScrollPane left = new UtilScrollPane(showTypePanel);
55         ValuePanel valuePanel = new ValuePanel(constant.getType(), constant, dialog, parent.
56             getGraphics());
56         UtilScrollPane right = new UtilScrollPane(valuePanel);
57         JSplitPane split = new JSplitPane(JSplitPane.VERTICAL_SPLIT, left, right);
58         split.setDividerLocation(0.25);
```

```

59         split.setResizeWeight(0.25);
60
61         dialog.getContentPane().add(split);
62         dialog.setModal(true);
63         dialog.setDefaultCloseOperation(JDialog.DO_NOTHING_ON_CLOSE);
64         dialog.addWindowListener(new WindowAdapter() {
65             @Override
66             public void windowClosing(WindowEvent e) {
67                 try {
68                     constant = valuePanel.getValue();
69                     DataPartConstant.this.setTitle(constant.getShortDescription());
70                     dialog.dispose();
71                 } catch (InvalidConstantException e1) {
72                     int confirm = JOptionPane.showOptionDialog(dialog,
73                         e1.getMessage() + ".\nClose without changing value?", "Exit
74                         confirmation",
75                         JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE, null, null,
76                         null);
77                     if (confirm == 0) {
78                         dialog.dispose();
79                     }
80                 });
81                 dialog.setVisible(true);
82             }
83
84             @Override
85             public boolean markDeletedIfAllowed() {
86                 return true;
87             }
88         }

```

LISTING E.79: gui/dataflow/DataPartConstant.java

```

1 package gui.dataflow;
2
3 import lang.type.Type;
4
5 public class Port {
6     private String name;
7     public String label;
8     private Type type;
9     public Port(String name, Type type) {
10         this.name = name;
11         this.label = name;
12         this.type = type;
13     }
14
15     public static Port[] clone(Port[] ps) {
16         Port[] ret = new Port[ps.length];
17         for (int i = 0; i < ps.length; i++) {
18             ret[i] = ps[i].clone();
19         }
20         return ret;

```

```
21     }
22
23     public Port clone() {
24         Port ret = new Port(name, type);
25         ret.label = label;
26         return ret;
27     }
28
29     public void setName(String name) {
30         this.name = name;
31     }
32
33     public String getName() {
34         return name;
35     }
36
37     public Type getType() {
38         return type;
39     }
40
41     public void setType(Type type) {
42         this.type = type;
43     }
44
45     @Override
46     public boolean equals(Object x) {
47         if (!(x instanceof Port))
48             return false;
49         Port p = (Port)x;
50         return p.name.equals(name);
51     }
52
53     @Override
54     public int hashCode() {
55         return name.hashCode();
56     }
57
58     @Override
59     public String toString() {
60         return "("+name+": "+type+)";
61     }
62 }
```

LISTING E.80: gui/dataflow/Port.java

```
1 package gui.dataflow.functional;
2
3 import lang.type.Type;
4
5 public interface IFunctional {
6     public boolean isParameterConnected(String param);
7     public boolean isResultConnected(String res);
8     public void disconnectInputIfExists(String name);
9     public void disconnectOutputIfExists(String name);
10    public void changeOutputPortType(String oldName, Type newType);
```

```
11     public void changeInputPortType(String oldName, Type newType);
12     public void changeInputPortName(String oldName, String newName);
13     public void changeOutputPortName(String oldName, String newName);
14 }
```

LISTING E.81: gui/dataflow/functional/IFunctional.java

```
1 package gui.dataflow;
2
3 import java.awt.BorderLayout;
4 import java.awt.Color;
5 import java.awt.Dimension;
6 import java.awt.FlowLayout;
7 import java.awt.Graphics;
8 import java.awt.GridBagLayout;
9 import java.awt.Window;
10 import java.awt.event.ActionEvent;
11 import java.awt.event.ActionListener;
12 import java.awt.event.WindowAdapter;
13 import java.awt.event.WindowEvent;
14 import java.util.function.Predicate;
15
16 import javax.swing.JButton;
17 import javax.swing.JDialog;
18 import javax.swing.JOptionPane;
19 import javax.swing.JPanel;
20 import javax.swing.JTextField;
21
22 import util.UtilButton;
23 import util.UtilLabel;
24 import util.UtilPrompt;
25 import util.UtilRegex;
26 import util.UtilScrollPane;
27 import gui.DataFlowPanel;
28 import gui.MainWindow;
29 import gui.ShowTypePanel;
30 import gui.TypeSelector;
31 import gui.controlflow.ControlPart;
32 import lang.type.Type;
33 import lang.type.TypeAutoChangeException;
34 import lang.type.TypeAutoInputListener;
35 import lang.type.TypeAutoOutputListener;
36 import lang.type.TypeUnknown;
37
38 public class DataPartResult extends DataPart implements TypeAutoInputListener,
39                                         TypeAutoOutputListener {
40
41     private final static Port[] outputPorts = {};
42     private String resultName;
43     private Type originalType;
44     private Type type;
45     private Predicate<DataPartResult> deleteEvent;
46     private DataFlowPanel containingPanel;
47     private boolean allowChangeType;
```

```

48     public DataPartResult(Type origType, Type type, String name, DataFlowPanel dfp,
49                           Graphics g, Predicate<DataPartResult> deleteEvent, boolean allowChangeType)
50                           throws InvalidDataPartException {
51         super(new Port[]{new Port("", type)}, outputPorts, name, "Output", dfp, g);
52         if(!UtilRegex.isIdentifier(name))
53             throw new InvalidDataPartException("Invalid output name '"+name+"'");
54         if (name.equals(ControlPart.errorEvent))
55             throw new InvalidDataPartException("Output name '"+ControlPart.errorEvent+"' is
56             reserved");
57         resultName = name;
58         this.originalType = origType;
59         this.type = type;
60         this.deleteEvent = deleteEvent;
61         containingPanel = getDataFlowPanel();
62         this.allowChangeType = allowChangeType;
63         //setInputArrowListeners(Arrays.asList(this));
64     }
65
66     public String getName() {
67         return resultName;
68     }
69
70     public Type getOriginalType() {
71         return originalType;
72     }
73
74     public Type getType() {
75         return type;
76     }
77
78     public void setType(Type t) {
79         type = t;
80     }
81
82     @Override
83     public boolean canFail() {
84         return false;
85     }
86
87     public void changeType(Type newType) {
88         changeType(newType, true);
89     }
90
91     public void changeType(Type newType, boolean disconect) {
92         changeInputPort(new Port("", type), new Port("", newType));
93         if (disconect)
94             disconnectInputTypeIfConnected("");
95         type = newType;
96         containingPanel.getParentTabbedPane().resultTypeChanged(containingPanel, this);
97     }
98
99     @Override
100    public void setTitle(String t) {
101        if (t.equals(ControlPart.errorEvent)) {
102            JOptionPane.showMessageDialog(getDataFlowPanel(),

```

```
102             "Output name '"+ControlPart.errorEvent+"' is reserved",
103             "Unable to change", JOptionPane.ERROR_MESSAGE);
104         return;
105     }
106     super.setTitle(t);
107 }
108
109     public void changeName(String newName) {
110         String oldName = resultName;
111         resultName = newName;
112         setTitle(resultName);
113         containingPanel.getParentTabbedPane().changeResultName(containingPanel, this, oldName);
114     }
115
116     @Override
117     public void doubleClick(MainWindow mainWindow, Window parent) {
118         ShowTypePanel showTypePanel = new ShowTypePanel(type, false);
119         showTypePanel.setLayout(new GridBagLayout());
120         showTypePanel.setBackground(Color.white);
121         JDialog dialog = new JDialog(parent);
122         dialog.setSize(new Dimension(500, 300));
123         dialog.setLocationRelativeTo(parent);
124         dialog.setModal(true);
125         dialog.getContentPane().add(showTypePanel);
126
127         JButton changeNameButton = new UtilButton("Change name");
128         JButton changeTypeButton = new UtilButton("Change type");
129         JPanel changePanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));
130         changePanel.add(changeNameButton);
131         changePanel.add(changeTypeButton);
132
133         JPanel namePanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));
134         namePanel.add(new UtilLabel(getName()));
135
136         changeNameButton.addActionListener(new ActionListener() {
137             @Override
138             public void actionPerformed(ActionEvent e) {
139                 //dialog.getContentPane().remove(namePanel);
140                 dialog.getContentPane().remove(changePanel);
141                 dialog.getContentPane().remove(showTypePanel);
142                 JPanel p = new JPanel(new GridBagLayout());
143                 JTextField field = new JTextField();
144                 UtilPrompt.setPrompt("New name", field);
145                 field.setPreferredSize(new Dimension(200, field.getPreferredSize().height));
146                 p.add(field);
147                 dialog.add(p);
148                 dialog.revalidate();
149                 dialog.repaint();
150                 dialog.setDefaultCloseOperation(JDialog.DO_NOTHING_ON_CLOSE);
151                 dialog.addWindowListener(new WindowAdapter() {
152                     @Override
153                     public void windowClosing(WindowEvent e) {
154                         String newName = field.getText();
155                         if (!UtilRegex.isIdentifier(newName)) {
156                             int confirm = JOptionPane.showOptionDialog(dialog,
```

```

157                     "Invalid output name '"+newName+"'.\nDo you want to discard
158                     the change?", 
159                     "Close confirmation", JOptionPane.YES_NO_OPTION,
160                     JOptionPane.QUESTION_MESSAGE, null, null, null);
160                     if (confirm == 0) {
161                         dialog.dispose();
162                         return;
163                     }
164                     return;
165                 }
166             try {
167                 getDataFlowPanel().validateAddDataPartResult(newName);
168             } catch (InvalidDataPartException e1) {
169                 int confirm = JOptionPane.showOptionDialog(dialog,
170                     e1.getMessage()+".\nDo you want to discard the change?", 
171                     "Close confirmation", JOptionPane.YES_NO_OPTION,
172                     JOptionPane.QUESTION_MESSAGE, null, null, null);
173                 if (confirm == 0) {
174                     dialog.dispose();
175                     return;
176                 }
177                 return;
178             }
179             changeName(newName);
180             dialog.dispose();
181         }
182     });
183 }
184 });
185
186 changeTypeButton.addActionListener(new ActionListener() {
187     @Override
188     public void actionPerformed(ActionEvent e) {
189         if (!allowChangeType) {
190             JOptionPane.showMessageDialog(dialog, "Cannot change type of this output",
191                     "Unable to change type", JOptionPane.ERROR_MESSAGE);
192             return;
193         }
194         if (containingPanel.getParentTabbedPane().isResultConnected(containingPanel,
195             DataPartResult.this)) {
196             String tabbedType = containingPanel.getParentTabbedPane() instanceof
197             DataFlowPanel ?
198                 "dataflow" : "controlflow";
199                 int confirm = JOptionPane.showOptionDialog(dialog,
200                     "Change will affect "+tabbedType+" '" +
201                     containingPanel.getParentTabbedPane().getTitle()+"'.\nDo you
202                     want to change?", 
203                     "Change type confirmation", JOptionPane.YES_NO_OPTION,
204                     JOptionPane.QUESTION_MESSAGE, null, null, null);
205                     if (confirm != 0)
206                         return;
207                     }
208             dialog.getContentPane().remove(changePanel);
209             dialog.getContentPane().remove(showTypePanel);
210             TypeSelector sel = new TypeSelector("New type", true);

```

```
208     dialog.add(new UtilScrollPane(sel));
209     dialog.revalidate();
210     dialog.repaint();
211     dialog.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
212     dialog.addWindowListener(new WindowAdapter() {
213         @Override
214         public void windowClosing(WindowEvent e) {
215             Type t = sel.getType();
216             /*if (t.equals(type)) {
217                 dialog.dispose();
218                 return;
219             }*/
220             if (!t.isConcrete(true)) {
221                 int confirm = JOptionPane.showOptionDialog(dialog,
222                     "Invalid type.\nDo you want to discard the change?",
223                     "Close confirmation", JOptionPane.YES_NO_OPTION,
224                     JOptionPane.QUESTION_MESSAGE, null, null, null);
225                 if (confirm == 0) {
226                     dialog.dispose();
227                     return;
228                 }
229                 return;
230             }
231             changeType(t);
232             originalType = t;
233             dialog.dispose();
234         }
235     });
236 });
237 });
238 dialog.getContentPane().add(namePanel, BorderLayout.NORTH);
239 dialog.getContentPane().add(changePanel, BorderLayout.SOUTH);
240 dialog.setVisible(true);
241 }
242
243 @Override
244 public boolean markDeletedIfAllowed() {
245     return deleteEvent.test(this);
246 }
247
248 @Override
249 public void autoTypeInputChanged(Type newType) throws TypeAutoChangeException {
250     changeType(newType, false);
251 }
252
253 @Override
254 public void autoTypeInputRemoved() {
255     if (autoTypeInputMayChange()) {
256         changeType(originalType, false);
257     }
258 }
259
260 @Override
261 public boolean autoTypeInputMayChange() {
```

```

262         return !getDataFlowPanel().getParentTabbedPane().isResultConnected(containingPanel, this)
263     ;
264 }
265 
266     @Override
267     public void autoTypeOutputRemoved() {
268         if (!isInputPortConnected(new Port("", new TypeUnknown("")))) &&
269             autoTypeInputMayChange()
270         changeType(originalType, false);
271     }

```

LISTING E.82: gui/dataflow/DataPartResult.java

```

1 package gui.dataflow;
2
3 public class DataPartBoolean {
4     public static String subtitle = "Boolean";
5 }

```

LISTING E.83: gui/dataflow/DataPartBoolean.java

```

1 package gui.dataflow;
2
3 import java.awt.Graphics;
4 import java.awt.Window;
5 import java.util.ArrayList;
6 import java.util.Arrays;
7 import java.util.List;
8
9 import lang.type.Type;
10 import lang.type.TypeAuto;
11 import lang.type.TypeAutoChangeEvent;
12 import lang.type.TypeAutoInputListener;
13 import lang.type.TypeAutoOutputListener;
14 import lang.type.TypeBool;
15 import lang.type.TypeList;
16 import lang.type.TypeNumber;
17 import lang.type.TypeUnknown;
18 import gui.DataFlowPanel;
19 import gui.MainWindow;
20
21 public class DataPartListOperation extends DataPart {
22     public final static String subtitle = "List";
23
24     public enum ListOperationType {APPEND, PREPEND, CONTAINS, AUGMENT, LENGTH, SUM, ALLTRUE,
25     ANYTRUE};
26
27     private final static Port[] listAutoPort = {new Port("", new TypeList(new TypeAuto()))};
28     private final static Port[] numberPort = {new Port("", new TypeNumber())};
29     private final static Port[] listNumberPort = {new Port("", new TypeList(new TypeNumber()))};
30     private final static Port[] listBooleanPort = {new Port("", new TypeList(new TypeBool()))};
31     private final static Port[] booleanPort = {new Port("", new TypeBool())};

```

```

32     private List<OperationType> listOperationType;
33     private List<Type> originalOperationTypes;
34     private List<Type> operationTypes;
35     private ArrayList<String> inputPortNames = new ArrayList<>();
36
37     private static Port[] getInputPorts(List<OperationType> type, List<Type> opTypes) {
38         switch (type) {
39             case LENGTH:
40                 return listAutoPort;
41             case SUM:
42                 return listNumberPort;
43             case ALLTRUE:
44                 return listBooleanPort;
45             case ANYTRUE:
46                 return listBooleanPort;
47             case APPEND:
48             case PREPEND:
49             case CONTAINS:
50                 return new Port[]{new Port("List", new TypeList(opTypes.get(0))), new Port("Item",
51                     opTypes.get(0))};
52             case AUGMENT:
53                 return new Port[]{new Port("List", new TypeList(opTypes.get(0))), new Port("Item",
54                     opTypes.get(1))};
55             default:
56                 throw new RuntimeException("unexpeceted code " + type);
57         }
58     }
59
60     private static Port[] getOutputPorts(List<OperationType> type, List<Type> opTypes) {
61         switch (type) {
62             case LENGTH:
63                 return numberPort;
64             case SUM:
65                 return numberPort;
66             case ALLTRUE:
67                 return booleanPort;
68             case ANYTRUE:
69                 return booleanPort;
70             case CONTAINS:
71                 return booleanPort;
72             case AUGMENT:
73                 Type t = new TypeList(DataPartUtil.getAugmentType(opTypes.get(0), "Left", opTypes.get
74 (1), "Right"));
75                 return new Port[]{new Port("", t)};
76             case APPEND:
77             case PREPEND:
78                 return new Port[]{new Port("", new TypeList(opTypes.get(0)))};
79             default:
80                 throw new RuntimeException("unexpeceted code " + type);
81         }
82     }
83
84     private static String getListOperationTitle(List<OperationType> type) {
85         switch (type) {
86             case LENGTH:
87

```

```
84         return "Length";
85     case SUM:
86         return "\u03A3";
87     case ALLTRUE:
88         return "All_true";
89     case ANYTRUE:
90         return "Any_true";
91     case CONTAINS:
92         return "Contains";
93     case AUGMENT:
94         return "Augment";
95     case APPEND:
96         return "Append";
97     case PREPEND:
98         return "Prepend";
99     default:
100         throw new RuntimeException("unexpeceted code " + type);
101    }
102 }
103
104 private static String getListOperationSubtitle(ListOperationType type) {
105     switch (type) {
106     case SUM:
107         return DataPartArithmetic.subtitle;
108     case ALLTRUE:
109     case ANYTRUE:
110         return DataPartBoolean.subtitle;
111     case APPEND:
112     case PREPEND:
113     case LENGTH:
114     case CONTAINS:
115     case AUGMENT:
116         return subtitle;
117     default:
118         throw new RuntimeException("unexpeceted code " + type);
119    }
120 }
121
122 public DataPartListOperation(ListOperationType type, DataFlowPanel dfp, Graphics g) {
123     this(type, null, null, dfp, g);
124 }
125
126 public DataPartListOperation(ListOperationType type, List<Type> origOpTypes, List<Type>
opTypes, DataFlowPanel dfp, Graphics g) {
127     super(getInputPorts(type, opTypes), getOutputPorts(type, opTypes), getListOperationTitle(
type),
128           getListOperationSubtitle(type), dfp, g);
129     listOperationType = type;
130     originalOperationTypes = new ArrayList<>();
131     operationTypes = new ArrayList<>();
132     if (opTypes != null) {
133         operationTypes.addAll(opTypes);
134         originalOperationTypes.addAll(origOpTypes);
135     }
136     for (Port p : getInputPorts(type, opTypes)) {
```

```
137         inputPortNames.add(p.getName());
138     }
139     setListeners();
140 }
141
142 private void setListeners() {
143     switch (listOperationType) {
144     case CONTAINS:
145         DoubleInputSingleTypeListener dsl0 = new DoubleInputSingleTypeListener(0);
146         DoubleInputSingleTypeListener dsl1 = new DoubleInputSingleTypeListener(1);
147         setInputArrowListeners(Arrays.asList(dsl0, dsl1));
148         break;
149     case LENGTH:
150         setInputArrowListeners(Arrays.asList(new LengthInputListener()));
151         break;
152     case APPEND:
153     case PREPEND:
154         DoubleInputSingleListOutputSingleTypeListener lst0 = new
155 DoubleInputSingleListOutputSingleTypeListener(0);
156         DoubleInputSingleListOutputSingleTypeListener lst1 = new
157 DoubleInputSingleListOutputSingleTypeListener(1);
158         setInputArrowListeners(Arrays.asList(lst0, lst1));
159         setOutputArrowListener("", new OutputListener(lst0, lst1));
160         break;
161     case AUGMENT:
162         DoubleInputDoubleTypeListener l0 = new DoubleInputDoubleTypeListener(0);
163         DoubleInputDoubleTypeListener l1 = new DoubleInputDoubleTypeListener(1);
164         setInputArrowListeners(Arrays.asList(l0, l1));
165         setOutputArrowListener("", new UnconditionOutputListener(l0, l1));
166         break;
167     case SUM:
168     case ALLTRUE:
169     case ANYTRUE:
170         break;
171     default:
172         throw new RuntimeException();
173     }
174 }
175
176 public List<Type> getOriginalOperationTypes() {
177     return new ArrayList<>(originalOperationTypes);
178 }
179
180 public List<Type> getOperationTypes() {
181     return new ArrayList<>(operationTypes);
182 }
183
184 public ListOperationType getListOperationType() {
185     return listOperationType;
186 }
187
188 @Override
189 public boolean canFail() {
190     return false;
```

```
190     }
191
192     @Override
193     public void doubleClick(MainWindow mainWindow, Window parent) {
194     }
195
196     @Override
197     public boolean markDeletedIfAllowed() {
198         return true;
199     }
200
201     private class LengthInputListener implements TypeAutoInputListener {
202         @Override
203         public void autoTypeInputChanged(Type newType) throws TypeAutoChangeException {
204             changeInputPortType("", newType);
205         }
206
207         @Override
208         public void autoTypeInputRemoved() {
209             changeInputPortType("", listAutoPort[0].getType());
210         }
211
212         @Override
213         public boolean autoTypeInputMayChange() {
214             return true;
215         }
216     }
217
218     private class DoubleInputSingleTypeListener implements TypeAutoInputListener {
219         private int index;
220
221         public DoubleInputSingleTypeListener(int i) {
222             index = i;
223         }
224
225         private Type getOtherType(Type t) {
226             if (index == 0)
227                 return ((TypeList)t).getChildType();
228             return new TypeList(t);
229         }
230
231         @Override
232         public void autoTypeInputChanged(Type newType) throws TypeAutoChangeException {
233             if (index == 0) {
234                 operationTypes.set(0, getOtherType(newType));
235                 changeInputPortType(inputPortNames.get(0), newType);
236                 changeInputPortType(inputPortNames.get(1), getOtherType(newType));
237             } else {
238                 operationTypes.set(0, newType);
239                 changeInputPortType(inputPortNames.get(0), getOtherType(newType));
240                 changeInputPortType(inputPortNames.get(1), newType);
241             }
242         }
243
244         @Override
```

```
245     public void autoTypeInputRemoved() {
246         if (autoTypeInputMayChange()) {
247             operationTypes.set(0, originalOperationTypes.get(0));
248             changeInputPortType(inputPortNames.get(0), new TypeList(originalOperationTypes.get
249             (0)));
250             changeInputPortType(inputPortNames.get(1), originalOperationTypes.get(0));
251         }
252     }
253
254     @Override
255     public boolean autoTypeInputMayChange() {
256         return !isInputPortConnected(new Port(inputPortNames.get((index+1)%2), new TypeUnknown
257         ("")));
258     }
259
260     private class DoubleInputSingleListOutputSingleTypeListener implements TypeAutoInputListener {
261         private int index;
262         DoubleInputSingleListOutputSingleTypeListener(int i) {
263             index = i;
264         }
265
266         @Override
267         public void autoTypeInputChanged(Type newType) throws TypeAutoChangeException {
268             if (index == 0) {
269                 operationTypes.set(0, ((TypeList)newType).getChildType());
270                 changeInputPortType(inputPortNames.get(0), newType);
271                 changeInputPortType(inputPortNames.get(1), ((TypeList)newType).getChildType());
272                 changeOutputPortType("", newType);
273             } else {
274                 operationTypes.set(0, newType);
275                 changeInputPortType(inputPortNames.get(0), new TypeList(newType));
276                 changeInputPortType(inputPortNames.get(1), newType);
277                 changeOutputPortType("", new TypeList(newType));
278             }
279         }
280     }
281
282     @Override
283     public void autoTypeInputRemoved() {
284         if (autoTypeInputMayChange()) {
285             operationTypes.set(0, originalOperationTypes.get(0));
286             changeInputPortType(inputPortNames.get(0), new TypeList(originalOperationTypes.get
287             (0)));
288             changeInputPortType(inputPortNames.get(1), originalOperationTypes.get(0));
289         }
290     }
291
292     @Override
293     public boolean autoTypeInputMayChange() {
294         return !isInputPortConnected(new Port(inputPortNames.get((index+1)%2), new TypeUnknown
295         ("")) &&
296             !isOutputPortConnected(new Port("", new TypeUnknown(""))));

```

```
296     }
297
298     }
299
300     private class OutputListener implements TypeAutoOutputListener {
301
302         private TypeAutoInputListener l0;
303         private TypeAutoInputListener l1;
304
305         public OutputListener(TypeAutoInputListener l0, TypeAutoInputListener l1) {
306             this.l0 = l0;
307             this.l1 = l1;
308         }
309
310         @Override
311         public void autoTypeOutputRemoved() {
312             if (l0.autoTypeInputMayChange() && l1.autoTypeInputMayChange()) {
313                 l0.autoTypeInputRemoved();
314             }
315         }
316     }
317
318     private class DoubleInputDoubleTypeListener implements TypeAutoInputListener {
319
320         private int index;
321
322         public DoubleInputDoubleTypeListener(int i) {
323             index = i;
324         }
325
326         @Override
327         public void autoTypeInputChanged(Type newType) throws TypeAutoChangeException {
328             if (index == 0) {
329                 operationTypes.set(index, ((TypeList)newType).getChildType());
330             } else {
331                 operationTypes.set(index, newType);
332             }
333             changeInputPortType(inputPortNames.get(index), newType);
334             changeOutputPortType("", getOutputPorts(listOperationType, operationTypes)[0].getType()
335             ());
336         }
337
338         @Override
339         public void autoTypeInputRemoved() {
340             if (autoTypeInputMayChange()) {
341                 operationTypes.set(index, originalOperationTypes.get(index));
342                 if (index == 0)
343                     changeInputPortType(inputPortNames.get(index), new TypeList(
344                     originalOperationTypes.get(index)));
345                 else
346                     changeInputPortType(inputPortNames.get(index), originalOperationTypes.get(
347                     index));
348                 changeOutputPortType("", getOutputPorts(listOperationType, operationTypes)[0].
349                 getType());
350             }
351         }
352     }
353 }
```

```

347
348     @Override
349     public boolean autoTypeInputMayChange() {
350         return !isOutputPortConnected(new Port("", new TypeUnknown("")));
351     }
352
353     public void autoTypeOutputRemoved() {
354         if (!isInputPortConnected(new Port(inputPortNames.get(index), new TypeUnknown(""))))
355             autoTypeInputRemoved();
356     }
357 }
358
359     private class UnconditionOutputListener implements TypeAutoOutputListener {
360
361         private DoubleInputDoubleTypeListener l0;
362         private DoubleInputDoubleTypeListener l1;
363
364         public UnconditionOutputListener(DoubleInputDoubleTypeListener l0,
365                                         DoubleInputDoubleTypeListener l1) {
365             this.l0 = l0;
366             this.l1 = l1;
367         }
368
369         @Override
370         public void autoTypeOutputRemoved() {
371             l0.autoTypeOutputRemoved();
372             l1.autoTypeOutputRemoved();
373         }
374     }
375 }
```

LISTING E.84: gui/dataflow/DataPartListOperation.java

```

1 package gui.dataflow;
2
3 import java.awt.BorderLayout;
4 import java.awt.FlowLayout;
5 import java.awt.Graphics;
6 import java.awt.Window;
7 import java.awt.event.WindowAdapter;
8 import java.awt.event.WindowEvent;
9 import java.util.ArrayList;
10 import java.util.List;
11
12 import javax.swing.JDialog;
13 import javax.swing.JOptionPane;
14 import javax.swing.JPanel;
15 import javax.swing.JTextArea;
16
17 import gui.DataFlowPanel;
18 import gui.MainWindow;
19 import lang.type.TypeBool;
20 import lang.type.TypeNumber;
21 import lang.type.TypeString;
22 import util.UtilLabel;
```

```
23 import util.UtilScrollPane;
24
25 public class DataPartStringFormat extends DataPart {
26
27     @SuppressWarnings("serial")
28     public static class FormatPanel extends JPanel {
29         private JTextArea text;
30
31         public FormatPanel(String format) {
32             setLayout(new BorderLayout());
33             JPanel labelPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 0, 0));
34             labelPanel.add(new UtilLabel("Enter format"));
35             add(labelPanel, BorderLayout.NORTH);
36
37             JPanel textPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));
38             text = new JTextArea(10, 25);
39             text.setText(format);
40             textPanel.add(new UtilScrollPane(text));
41
42             add(textPanel);
43         }
44
45         public FormatPanel() {
46             this("");
47         }
48
49         public String getFormat() {
50             return text.getText();
51         }
52     }
53
54     private static final Port[] result = {new Port("!0", new TypeString())};
55
56     private static Port[] getInputPorts(String format) throws InvalidDataPartException {
57         int index = 1;
58         ArrayList<Port> ports = new ArrayList<>();
59         for (int i = 0; i < format.length(); i++) {
60             char c = format.charAt(i);
61             if (c == '%') {
62                 if (i == format.length()-1) {
63                     throw new InvalidDataPartException("Expected character after '%' in format");
64                 }
65                 c = format.charAt(++i);
66                 switch (c) {
67                     case 's':
68                         ports.add(new Port("String" + index++, new TypeString()));
69                         break;
70                     case 'n':
71                         ports.add(new Port("Number" + index++, new TypeNumber()));
72                         break;
73                     case 'b':
74                         ports.add(new Port("Boolean" + index++, new TypeBool()));
75                         break;
76                     case '%':
77                         break;
78                 }
79             }
80         }
81         return ports.toArray(new Port[ports.size()]);
82     }
83 }
```

```
78         default:
79             throw new InvalidDataPartException("Invalid format code '%"+c+"'");
80     }
81 }
82 }
83
84 Port[] ret = new Port[ports.size()];
85 ports.toArray(ret);
86 return ret;
87 }
88
89 private String format;
90
91 public DataPartStringFormat(String format, DataFlowPanel dfp, Graphics g)
92     throws InvalidDataPartException {
93     super(getInputPorts(format), result, "Format: "+format, "String", dfp, g);
94     this.format = format;
95 }
96
97 public String getFormat() {
98     return format;
99 }
100
101 @Override
102 public boolean canFail() {
103     return false;
104 }
105
106 @Override
107 public void doubleClick(MainWindow mainWindow, Window parent) {
108     FormatPanel panel = new FormatPanel(format);
109
110     JDialog dialog = new JDialog(parent);
111     dialog.setSize(500, 300);
112     dialog.setLocationRelativeTo(parent);
113     dialog.setModal(true);
114     dialog.setDefaultCloseOperation(JDialog.DO_NOTHING_ON_CLOSE);
115
116     dialog.addWindowListener(new WindowAdapter() {
117         @Override
118         public void windowClosing(WindowEvent e) {
119             String newFmt = panel.getFormat();
120             try {
121                 Port[] newPorts = getInputPorts(newFmt);
122                 List<InputArrow> as = getInputArrows();
123                 format = newFmt;
124                 setTitle("Format: "+format);
125                 block: {
126                     if (newPorts.length != as.size())
127                         break block;
128                     for (int i = 0; i < as.size(); i++) {
129                         InputArrow a = as.get(i);
130                         Port p = newPorts[i];
131                         if (!p.getType().equals(a.getType()))
132                             break block;
133                 }
134             }
135         }
136     });
137 }
```

```
133             }
134             dialog.dispose();
135             return;
136         }
137         removeAllInputPorts();
138         addAllInputPorts(newPorts);
139         dialog.dispose();
140     } catch (InvalidDataPartException ex) {
141         int confirm = JOptionPane.showOptionDialog(parent,
142             ex.getMessage()+"\nDo you want to discard the change?",
143             "Close confirmation", JOptionPane.YES_NO_OPTION,
144             JOptionPane.QUESTION_MESSAGE, null, null, null);
145         if (confirm == 0) {
146             dialog.dispose();
147         }
148     }
149 }
150 });
151 dialog.getContentPane().add(panel);
152 dialog.setVisible(true);
153 }
154 }
155
156 @Override
157 public boolean markDeletedIfAllowed() {
158     return true;
159 }
160
161 }
```

LISTING E.85: gui/dataflow/DataPartStringFormat.java

```
1 package gui.dataflow;
2
3 import java.awt.BasicStroke;
4 import java.awt.Graphics2D;
5 import java.awt.Point;
6 import java.awt.Rectangle;
7 import java.awt.Stroke;
8 import java.awt.geom.Line2D;
9 import java.awt.geom.Point2D;
10
11 public class LineConnectPart {
12     private DataPart.InputArrow inputArrow;
13     private DataPart.OutputArrow outputArrow;
14
15     public LineConnectPart(DataPart.InputArrow inputArrow, DataPart.OutputArrow outputArrow) {
16         this.inputArrow = inputArrow;
17         this.outputArrow = outputArrow;
18     }
19     public DataPart.InputArrow getInputArrow(){
20         return inputArrow;
21     }
22     public DataPart.OutputArrow getOutputArrow(){
23         return outputArrow;
24     }
25 }
```

```
24     }
25     public Line2D getLine() {
26         return new Line2D.Double(inputArrow.getOvalCenter(), outputArrow.getOvalCenter());
27     }
28     public Rectangle getBoundingRect() {
29         Rectangle r1 = getLine().getBounds();
30         return new Rectangle(r1.x-3,r1.y-3,r1.width+6,r1.height+6);
31     }
32
33     public boolean sameLine(LineConnectPart y) {
34         if (y == null)
35             return false;
36         return inputArrow == y.inputArrow && outputArrow == y.outputArrow;
37     }
38
39     private final int HIT_BOX_SIZE = 6;
40     public boolean intersects(Point p){
41         Line2D line = getLine();
42         int boxX = p.x - HIT_BOX_SIZE/2;
43         int boxY = p.y - HIT_BOX_SIZE/2;
44
45         int width = HIT_BOX_SIZE;
46         int height = HIT_BOX_SIZE;
47
48         return line.intersects(boxX, boxY, width, height);
49     }
50
51     public static class Temporary {
52         Line2D line;
53         private Temporary() {}
54         public void paint(Graphics2D g2) {
55             Stroke s = g2.getStroke();
56             g2.setStroke(new BasicStroke(2));
57             g2.draw(line);
58             g2.setStroke(s);
59         }
60
61         public Rectangle getBoundingRectangle() {
62             Rectangle r = line.getBounds();
63             return new Rectangle((int)r.getX()-6, (int)r.getY()-6,(int)r.getWidth()+12,(int)r.
64             getHeight()+12);
65         }
66
67         public Rectangle setEndPoint(Point newEnd) {
68             Rectangle ret = getBoundingRectangle();
69             line.setLine(line.getPt1(), newEnd);
70             return ret;
71         }
72
73         public Point2D getStartPoint() {
74             return line.getPt1();
75         }
76
77     public static Temporary makeTemporary(Point start, Point end) {
```

```
78     Temporary ret = new Temporary();
79     ret.line = new Line2D.Double(start, end);
80     return ret;
81 }
82 }
```

LISTING E.86: gui/dataflow/LineConnectPart.java

```
1 package gui.dataflow;
2
3 import java.awt.BorderLayout;
4 import java.awt.Dimension;
5 import java.awt.FlowLayout;
6 import java.awt.Graphics;
7 import java.awt.GridBagLayout;
8 import java.awt.Window;
9 import java.awt.event.ActionEvent;
10 import java.awt.event.ActionListener;
11 import java.awt.event.ItemEvent;
12 import java.awt.event.ItemListener;
13 import java.awt.event.WindowAdapter;
14 import java.awt.event.WindowEvent;
15 import java.util.Arrays;
16 import java.util.function.BiPredicate;
17
18 import javax.swing.Box;
19 import javax.swing.BoxLayout;
20 import javax.swing.JComboBox;
21 import javax.swing.JDialog;
22 import javax.swing.JLabel;
23 import javax.swing.JOptionPane;
24 import javax.swing.JPanel;
25 import javax.swing.JSplitPane;
26 import javax.swing.JTextField;
27
28 import gui.ConstantDefiner;
29 import gui.DataFlowPanel;
30 import gui.TabbedPane;
31 import gui.TabbedPane.InvalidTitleException;
32 import gui.TypeSelector;
33 import lang.constant.Constant;
34 import lang.constant.InvalidConstantException;
35 import lang.type.TypeCode;
36 import lang.type.TypeNumber;
37 import lang.type.TypeRecord;
38 import util.UtilFont;
39 import util.UtilPrompt;
40 import util.UtilScrollPane;
41
42 @SuppressWarnings("serial")
43 public class DataPartDialog extends JDialog implements ActionListener {
44     private final static int
45             FUNCTIONAL_INDEX = 0,
46             SUBDATAFLOW_INDEX = 1,
47             CONSTANT_INDEX = 2,
```

```
48     TYPE_CAST_INDEX = 3,
49     RECORD_CONSTRUCTOR =4,
50     SERVICE_INDEX = 5,
51     PARAMETER_INDEX = 6,
52     RESULT_INDEX = 7,
53     SINK_INDEX = 8;
54 
55     private final static String[] comboBoxOptions =
56         {"Functional", "Subdataflow", "Constant", "Type cast", "Record constructor", "Service", "Input", "Output", "Sink"};
57 
58     private final static int
59         FUN_ARITHMETIC_INDEX = 0,
60         FUN_STRING_OPERATION_INDEX = 1,
61         FUN_COMPARISON_INDEX = 2,
62         FUN_LIST_OPERATION_INDEX = 3,
63         FUN_BOOLEAN_INDEX = 4;
64 
65     private final static String[] functionalComboBoxOptions =
66         {"Arithmetic", "String", "Comparison", "List", "Boolean"};
67 
68     private JComboBox<String> comboBox;
69     private UtilScrollPane scrollPane;
70     private DialogPanel dialogPanel;
71     private DataFlowPanel dataFlowPanel;
72     private JPanel currentFunctionalPanel;
73     private Window parentWindow;
74 
75     public DataPartDialog(Window parent, DataFlowPanel dataFlowPanel) {
76         super(parent);
77         parentWindow = parent;
78         this.dataFlowPanel = dataFlowPanel;
79 
80         setSize(600, 500);
81         setLocationRelativeTo(parent);
82         setModal(true);
83         setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
84         this.addWindowListener(new WindowAdapter() {
85             @Override
86             public void windowClosing(WindowEvent e) {
87                 if (dialogPanel == null) {
88                     DataPartDialog.this.dispose();
89                     return;
90                 }
91                 dialogPanel.applyOperationExit();
92             }
93         });
94         setVisible(true);
95     }
96 
97     private void setComboBox() {
98         comboBox = new JComboBox<>(comboBoxOptions);
99         comboBox.setSelectedItem(null);
100        comboBox.setFont(UtilFont.textFont);
101        comboBox.addActionListener(this);
102    }
103 }
```

```
102
103     JPanel inner = new JPanel();
104     inner.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
105     JLabel label = new JLabel("Component type");
106     label.setFont(UtilFont.textFont);
107     inner.add(label);
108     inner.add(comboBox);
109     JPanel outer = new JPanel(new GridBagLayout());
110     outer.add(inner);
111
112     getContentPane().add(outer, BorderLayout.NORTH);
113 }
114
115 @Override
116 public void actionPerformed(ActionEvent e) {
117     switch (comboBox.getSelectedIndex()) {
118         case FUNCTIONAL_INDEX:
119             setFunctionalPanel();
120             break;
121         case SUBDATAFLOW_INDEX:
122             setSubdataflowPanel();
123             break;
124         case CONSTANT_INDEX:
125             setConstantPanel(parentWindow.getGraphics());
126             break;
127         case TYPE_CAST_INDEX:
128             setTypeCastPanel();
129             break;
130         case PARAMETER_INDEX:
131             setParameterPanel();
132             break;
133         case RESULT_INDEX:
134             setResultPanel();
135             break;
136         case RECORD_CONSTRUCTOR:
137             setRecordConstructorPanel();
138             break;
139         case SERVICE_INDEX:
140             setServicePanel();
141             break;
142         case SINK_INDEX:
143             setSinkPanel();
144             break;
145     default:
146         break;
147     }
148 }
149
150 private void resetScrollPane(JPanel p) {
151     if (scrollPane != null)
152         getContentPane().remove(scrollPane);
153     scrollPane = new UtilScrollPane(p);
154     getContentPane().add(scrollPane);
155     revalidate();
156     repaint();
```

```

157     }
158
159     private JPanel getFunctionalComparisonPanel(){
160         JPanel panel = new JPanel(new BorderLayout());
161         JPanel cboxPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));
162         String[] options = {"<", ">", "\u2264", "\u2265", "=" , "\u2260"};
163         JComboBox<String> cbox = new JComboBox<>(options);
164         cbox.setFont(UtilFont.textFont);
165         cbox.setSelectedItem(null);
166         cboxPanel.add(cbox);
167         panel.add(cboxPanel, BorderLayout.NORTH);
168
169         TypeSelector sel = new TypeSelector("Input type", true);
170         sel.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
171
172         cbox.addActionListener(new ActionListener() {
173             @Override
174             public void actionPerformed(ActionEvent e) {
175                 int index = cbox.getSelectedIndex();
176                 switch (index) {
177                     case 0:
178                         panel.remove(sel);
179                         panel.revalidate();
180                         panel.repaint();
181                         dialogPanel = new DialogPanel() {
182                             @Override
183                             public void applyOperationExit() {
184                                 try {
185                                     dataFlowPanel.addNewDataPartComparison(
186                                         DataPartComparison.ComparisonType.LESS,
187                                         new TypeNumber(), new TypeNumber());
188                                     DataPartDialog.this.dispose();
189                                 } catch (InvalidDataPartException e) {
190                                     throw new RuntimeException();
191                                 }
192                             }
193                         };
194                         break;
195                     case 1:
196                         panel.remove(sel);
197                         panel.revalidate();
198                         panel.repaint();
199                         dialogPanel = new DialogPanel() {
200                             @Override
201                             public void applyOperationExit() {
202                                 try {
203                                     dataFlowPanel.addNewDataPartComparison(
204                                         DataPartComparison.ComparisonType.GREATER,
205                                         new TypeNumber(), new TypeNumber());
206                                     DataPartDialog.this.dispose();
207                                 } catch (InvalidDataPartException e) {
208                                     throw new RuntimeException();
209                                 }
210                             }
211                         };
}

```

```
212     };
213     break;
214 case 2:
215     panel.remove(sel);
216     panel.revalidate();
217     panel.repaint();
218     dialogPanel = new DialogPanel() {
219         @Override
220         public void applyOperationExit() {
221             try {
222                 dataFlowPanel.addNewDataPartComparison(
223                     DataPartComparison.ComparisonType.LESSEQUAL,
224                     new TypeNumber(), new TypeNumber());
225                 DataPartDialog.this.dispose();
226             } catch (InvalidDataPartException e) {
227                 throw new RuntimeException();
228             }
229         }
230     };
231     break;
232 case 3:
233     panel.remove(sel);
234     panel.revalidate();
235     panel.repaint();
236     dialogPanel = new DialogPanel() {
237         @Override
238         public void applyOperationExit() {
239             try {
240                 dataFlowPanel.addNewDataPartComparison(
241                     DataPartComparison.ComparisonType.GREATEREQUAL,
242                     new TypeNumber(), new TypeNumber());
243                 DataPartDialog.this.dispose();
244             } catch (InvalidDataPartException e) {
245                 throw new RuntimeException();
246             }
247         }
248     };
249     break;
250 case 4:
251     panel.remove(sel);
252     panel.add(sel);
253     panel.revalidate();
254     panel.repaint();
255     dialogPanel = new DialogPanel() {
256         @Override
257         public void applyOperationExit() {
258             try {
259                 dataFlowPanel.addNewDataPartComparison(
260                     DataPartComparison.ComparisonType.EQUAL,
261                     sel.getType(), sel.getType());
262                 DataPartDialog.this.dispose();
263             } catch (InvalidDataPartException e) {
264                 int confirm = JOptionPane.showOptionDialog(DataPartDialog.this,
```

```
267                         e.getMessage() + ".\nDo you want to discard the comparison  
268                         ?",  
269                         "Invalid component", JOptionPane.YES_NO_OPTION,  
270                         JOptionPane.QUESTION_MESSAGE, null, null, null);  
271                         if (confirm == 0) {  
272                             DataPartDialog.this.dispose();  
273                         }  
274                     }  
275                 }  
276             };  
277             break;  
278         case 5:  
279             panel.remove(sel);  
280             panel.add(sel);  
281             panel.revalidate();  
282             panel.repaint();  
283             dialogPanel = new DialogPanel() {  
284                 @Override  
285                 public void applyOperationExit() {  
286                     try {  
287                         dataFlowPanel.addNewDataPartComparison(  
288                             DataPartComparison.ComparisonType.NOTEQUAL,  
289                             sel.getType(), sel.getType());  
290                         DataPartDialog.this.dispose();  
291                     } catch (InvalidDataPartException e) {  
292                         int confirm = JOptionPane.showOptionDialog(DataPartDialog.this,  
293                             e.getMessage() + ".\nDo you want to discard the comparison  
294                             ?",  
295                             "Invalid component", JOptionPane.YES_NO_OPTION,  
296                             JOptionPane.QUESTION_MESSAGE, null, null, null);  
297                             if (confirm == 0) {  
298                                 DataPartDialog.this.dispose();  
299                             }  
300                         }  
301                     }  
302                 };  
303                 break;  
304             default:  
305                 assert false;  
306             }  
307         }  
308     });  
309  
310     return panel;  
311 }  
312  
313 private JPanel getFunctionalBooleanPanel() {  
314     JPanel panel = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));  
315     // And, Or, Not  
316     String[] options = {"\u2227", "\u2228", "\u00AC", "All_true", "Any_true"};  
317     JComboBox<String> cbox = new JComboBox<>(options);  
318     cbox.setFont(UtilFont.textFont);  
319     cbox.setSelectedItem(null);
```

```
320     panel.add(cbox);
321
322     cbox.addActionListener(new ActionListener() {
323         @Override
324         public void actionPerformed(ActionEvent e) {
325             int index = cbox.getSelectedIndex();
326             switch (index) {
327                 case 0:
328                     dialogPanel = new DialogPanel() {
329                         @Override
330                         public void applyOperationExit() {
331                             dataFlowPanel.addNewDataPartArithmetic(DataPartArithmetic.
332                                 ArithmeticType.AND);
333                             DataPartDialog.this.dispose();
334                         }
335                     };
336                     break;
337                 case 1:
338                     dialogPanel = new DialogPanel() {
339                         @Override
340                         public void applyOperationExit() {
341                             dataFlowPanel.addNewDataPartArithmetic(DataPartArithmetic.
342                                 ArithmeticType.OR);
343                             DataPartDialog.this.dispose();
344                         }
345                     };
346                     break;
347                 case 2:
348                     dialogPanel = new DialogPanel() {
349                         @Override
350                         public void applyOperationExit() {
351                             dataFlowPanel.addNewDataPartArithmetic(DataPartArithmetic.
352                                 ArithmeticType.NOT);
353                             DataPartDialog.this.dispose();
354                         }
355                     };
356                     break;
357                 case 3:
358                     dialogPanel = new DialogPanel() {
359                         @Override
360                         public void applyOperationExit() {
361                             dataFlowPanel.addNewDataPartListOperation(DataPartListOperation.
362                                 ListOperationType.ALLTRUE);
363                             DataPartDialog.this.dispose();
364                         }
365                     };
366                     break;
367                 case 4:
368                     dialogPanel = new DialogPanel() {
369                         @Override
370                         public void applyOperationExit() {
371                             dataFlowPanel.addNewDataPartListOperation(DataPartListOperation.
372                                 ListOperationType.ANYTRUE);
373                             DataPartDialog.this.dispose();
374                         }
375                     };
376             }
377         }
378     }
```

```
370         };
371         break;
372     default:
373         assert false;
374     }
375 }
376 });
377
378 return panel;
379 }
380
381 private JPanel getFunctionalArithmeticPanel() {
382     JPanel panel = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));
383     // plus, minus, mul, div, Negate, Sum
384     String[] options = {"\u002B", "\u2212", "\u00D7", "\u00F7", "Negate", "\u03A3"};
385     JComboBox<String> cbox = new JComboBox<>(options);
386     cbox.setFont(UtilFont.textFont);
387     cbox.setSelectedItem(null);
388     panel.add(cbox);
389
390     cbox.addActionListener(new ActionListener() {
391         @Override
392         public void actionPerformed(ActionEvent e) {
393             int index = cbox.getSelectedIndex();
394             switch (index) {
395                 case 0:
396                     dialogPanel = new DialogPanel() {
397                         @Override
398                         public void applyOperationExit() {
399                             dataFlowPanel.addNewDataPartArithmetic(DataPartArithmetic.
400                                 ArithmeticType.PLUS);
401                             DataPartDialog.this.dispose();
402                         }
403                     };
404                     break;
405                 case 1:
406                     dialogPanel = new DialogPanel() {
407                         @Override
408                         public void applyOperationExit() {
409                             dataFlowPanel.addNewDataPartArithmetic(DataPartArithmetic.
410                                 ArithmeticType_MINUS);
411                             DataPartDialog.this.dispose();
412                         }
413                     };
414                     break;
415                 case 2:
416                     dialogPanel = new DialogPanel() {
417                         @Override
418                         public void applyOperationExit() {
419                             dataFlowPanel.addNewDataPartArithmetic(DataPartArithmetic.
420                                 ArithmeticType.MUL);
421                             DataPartDialog.this.dispose();
422                         }
423                     };
424                     break;
```

```
422         case 3:
423             dialogPanel = new DialogPanel() {
424                 @Override
425                 public void applyOperationExit() {
426                     dataFlowPanel.addNewDataPartArithmetic(DataPartArithmetic.
427                         ArithmeticType.DIV);
428                     DataPartDialog.this.dispose();
429                 }
430             };
431             break;
432         case 4:
433             dialogPanel = new DialogPanel() {
434                 @Override
435                 public void applyOperationExit() {
436                     dataFlowPanel.addNewDataPartArithmetic(DataPartArithmetic.
437                         ArithmeticType.NEGATE);
438                     DataPartDialog.this.dispose();
439                 }
440             };
441             break;
442         case 5:
443             dialogPanel = new DialogPanel() {
444                 @Override
445                 public void applyOperationExit() {
446                     dataFlowPanel.addNewDataPartListOperation(DataPartListOperation.
447                         ListOperationType.SUM);
448                     DataPartDialog.this.dispose();
449                 }
450             };
451             break;
452         default:
453             assert false;
454     });
455     return panel;
456 }
457
458 private JPanel getFunctionalStringPanel() {
459     JPanel panel = new JPanel(new BorderLayout());
460     String[] options = {"Concatenate", "Contains", "Length", "Format"};
461     JComboBox<String> cbox = new JComboBox<>(options);
462     cbox.setFont(UtilFont.textField);
463     cbox.setSelectedItem(null);
464     JPanel boxPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));
465     boxPanel.add(cbox);
466     panel.add(boxPanel, BorderLayout.NORTH);
467
468     DataPartStringFormat.FormatPanel formatPanel = new DataPartStringFormat.FormatPanel();
469
470     cbox.addActionListener(new ActionListener() {
471         @Override
472         public void actionPerformed(ActionEvent e) {
473             panel.remove(formatPanel);
```

```
474     int index = cbox.getSelectedIndex();
475     switch (index) {
476     case 0:
477         dialogPanel = new DialogPanel() {
478             @Override
479             public void applyOperationExit() {
480                 dataFlowPanel.addNewDataPartStringOperation(DataPartStringOperation.
481                     StringOperationType.CONCATENATE);
482                     DataPartDialog.this.dispose();
483                 }
484             break;
485         case 1:
486         dialogPanel = new DialogPanel() {
487             @Override
488             public void applyOperationExit() {
489                 dataFlowPanel.addNewDataPartStringOperation(DataPartStringOperation.
490                     StringOperationType.CONTAINS);
491                     DataPartDialog.this.dispose();
492                 }
493             break;
494         case 2:
495         dialogPanel = new DialogPanel() {
496             @Override
497             public void applyOperationExit() {
498                 dataFlowPanel.addNewDataPartStringOperation(DataPartStringOperation.
499                     StringOperationType.LENGTH);
500                     DataPartDialog.this.dispose();
501                 }
502             break;
503         case 3:
504             panel.add(formatPanel);
505             dialogPanel = new DialogPanel() {
506                 @Override
507                 public void applyOperationExit() {
508                     try {
509                         dataFlowPanel.addNewDataPartStringFormat(formatPanel.getFormat());
510                         DataPartDialog.this.dispose();
511                     } catch (InvalidDataPartException e) {
512                         int confirm = JOptionPane.showOptionDialog(DataPartDialog.this,
513                             e.getMessage() + ".\nDo you want to discard the string
514                             operation?", 
515                             "Invalid format", JOptionPane.YES_NO_OPTION,
516                             JOptionPane.QUESTION_MESSAGE, null, null, null);
517                         if (confirm == 0) {
518                             DataPartDialog.this.dispose();
519                         }
520                     }
521                 };
522             break;
523         default:
524             assert false;
```

```
525         }
526
527         panel.revalidate();
528         panel.repaint();
529     }
530 );
531
532     return panel;
533 }
534
535     private JPanel getFunctionalListPanel() {
536         JPanel panel = new JPanel(new BorderLayout());
537         String[] options = {"Map", "Filter", "For_each", "Join", "Accumulate", "Length", "Append",
538 "Prepend", "Augment", "Contains"};
539         JComboBox<String> cbox = new JComboBox<>(options);
540         cbox.setFont(UtilFont.textFont);
541         cbox.setSelectedItem(null);
542         JPanel cboxPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));
543 //cboxPanel.setBorder(BorderFactory.createMatteBorder(0, 0, 1, 0, Color.black));
544         cboxPanel.add(cbox);
545         panel.add(cboxPanel, BorderLayout.NORTH);
546
547         JPanel titlePanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));
548         JTextField titleField = new JTextField();
549         UtilPrompt.setPrompt("Subdataflow title", titleField);
550         titleField.setPreferredSize(new Dimension(200, titleField.getPreferredSize().height));
551         titleField.setFont(UtilFont.textFont);
552         titlePanel.add(titleField);
553
554         TypeSelector opTypePanel = new TypeSelector("List type", true);
555         JPanel opPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));
556         opPanel.add(opTypePanel);
557
558         TypeSelector opTypePanel1 = new TypeSelector("List type", true);
559         TypeSelector opTypePanel2 = new TypeSelector("Item type", true);
560         JSplitPane opTypeSplitPanel = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
561             opTypePanel1, opTypePanel2);
562         opTypeSplitPanel.setDividerLocation(0.5);
563         opTypeSplitPanel.setResizeWeight(0.5);
564         JPanel opSplitPanel = new JPanel(new BorderLayout(5, 5));
565         opSplitPanel.add(opTypeSplitPanel);
566
567         cbox.addActionListener(new ActionListener() {
568             @Override
569             public void actionPerformed(ActionEvent e) {
570                 panel.remove(titlePanel);
571                 panel.remove(opPanel);
572                 panel.remove(opSplitPanel);
573
574                 if (cbox.getSelectedItem().equals("Map")) {
575                     panel.add(titlePanel);
576                     dialogPanel = new DialogPanel() {
577                         @Override
578                         public void applyOperationExit() {
579                             try {
```

```
579                     dataFlowPanel.addNewDataPartFunctional(
580                         DataPartFunctional.FunctionalType.MAP, titleField.getText
581                     );
582                 DataPartDialog.this.dispose();
583             } catch (InvalidTitleException | InvalidDataPartException e) {
584                 int confirm = JOptionPane.showOptionDialog(DataPartDialog.this,
585                     e.getMessage() + ".\nDo you want to discard the list
586                     operation?", "
587                     "Invalid component", JOptionPane.YES_NO_OPTION,
588                     JOptionPane.QUESTION_MESSAGE, null, null, null);
589                 if (confirm == 0) {
590                     DataPartDialog.this.dispose();
591                 }
592             }
593         } else if (cbox.getSelectedItem().equals("Filter")) {
594             panel.add(titlePanel);
595             dialogPanel = new DialogPanel() {
596                 @Override
597                 public void applyOperationExit() {
598                     try {
599                         dataFlowPanel.addNewDataPartFunctional(
600                             DataPartFunctional.FunctionalType.FILTER, titleField.
601                         getText());
602                         DataPartDialog.this.dispose();
603                     } catch (InvalidTitleException | InvalidDataPartException e) {
604                         int confirm = JOptionPane.showOptionDialog(DataPartDialog.this,
605                             e.getMessage() + ".\nDo you want to discard the list
606                             operation?", "
607                             "Invalid component", JOptionPane.YES_NO_OPTION,
608                             JOptionPane.QUESTION_MESSAGE, null, null, null);
609                         if (confirm == 0) {
610                             DataPartDialog.this.dispose();
611                         }
612                     }
613                 };
614             } else if (cbox.getSelectedItem().equals("Join")) {
615                 panel.add(titlePanel);
616                 dialogPanel = new DialogPanel() {
617                     @Override
618                     public void applyOperationExit() {
619                         try {
620                             dataFlowPanel.addNewDataPartFunctional(
621                                 DataPartFunctional.FunctionalType.JOIN, titleField.getText
622                             ());
623                             DataPartDialog.this.dispose();
624                         } catch (InvalidTitleException | InvalidDataPartException e) {
625                             int confirm = JOptionPane.showOptionDialog(DataPartDialog.this,
626                             e.getMessage() + ".\nDo you want to discard the list
627                             operation?", "
628                             "Invalid component", JOptionPane.YES_NO_OPTION,
629                             JOptionPane.QUESTION_MESSAGE, null, null, null);
630                         if (confirm == 0) {
```

```
628                     DataPartDialog.this.dispose();
629                 }
630             }
631         }
632     };
633 } else if (cbox.getSelectedItem().equals("Accumulate")) {
634     panel.add(titlePanel);
635     dialogPanel = new DialogPanel() {
636         @Override
637         public void applyOperationExit() {
638             try {
639                 dataFlowPanel.addNewDataPartFunctional(
640                     DataPartFunctional.FunctionalType.ACUMULATE, titleField.
641                     getText());
642                 DataPartDialog.this.dispose();
643             } catch (InvalidTitleException | InvalidDataPartException e) {
644                 int confirm = JOptionPane.showOptionDialog(DataPartDialog.this,
645                     e.getMessage() + ".\nDo you want to discard the list
646                     operation?", "
647                     "Invalid component", JOptionPane.YES_NO_OPTION,
648                     JOptionPane.QUESTION_MESSAGE, null, null, null);
649                 if (confirm == 0) {
650                     DataPartDialog.this.dispose();
651                 }
652             }
653         };
654     } else if (cbox.getSelectedItem().equals("For_each")) {
655         panel.add(titlePanel);
656         dialogPanel = new DialogPanel() {
657             @Override
658             public void applyOperationExit() {
659                 try {
660                     dataFlowPanel.addNewDataPartFunctional(
661                         DataPartFunctional.FunctionalType.FOREACH, titleField.
662                         getText());
663                     DataPartDialog.this.dispose();
664                 } catch (InvalidTitleException | InvalidDataPartException e) {
665                     int confirm = JOptionPane.showOptionDialog(DataPartDialog.this,
666                     e.getMessage() + ".\nDo you want to discard the list
667                     operation?", "
668                     "Invalid component", JOptionPane.YES_NO_OPTION,
669                     JOptionPane.QUESTION_MESSAGE, null, null, null);
670                 if (confirm == 0) {
671                     DataPartDialog.this.dispose();
672                 }
673             };
674     } else if (cbox.getSelectedItem().equals("Contains")) {
675         panel.add(opPanel);
676         dialogPanel = new DialogPanel() {
677             @Override
678             public void applyOperationExit() {
lang.type.Type t = opTypePanel.getType();
```

```
679         if (t.isConcrete(true)) {
680             dataFlowPanel.addNewDataPartListOperation(DataPartListOperation.
681                 ListOperationType.CONTAINS, Arrays.asList(t), Arrays.asList(t));
682             DataPartDialog.this.dispose();
683         } else {
684             int confirm = JOptionPane.showOptionDialog(DataPartDialog.this,
685                 "Invalid type.\nDo you want to discard the list operation?
686                 ",
687                 "Invalid component", JOptionPane.YES_NO_OPTION,
688                 JOptionPane.QUESTION_MESSAGE, null, null, null);
689             if (confirm == 0) {
690                 DataPartDialog.this.dispose();
691             }
692         }
693     };
694     } else if (cbox.getSelectedItem().equals("Append")) {
695         panel.add(opPanel);
696         dialogPanel = new DialogPanel() {
697             @Override
698             public void applyOperationExit() {
699                 lang.type.Type t = opTypePanel.getType();
700                 if (t.isConcrete(true)) {
701                     dataFlowPanel.addNewDataPartListOperation(DataPartListOperation.
702                         ListOperationType.APPEND, Arrays.asList(t), Arrays.asList(t));
703                     DataPartDialog.this.dispose();
704                 } else {
705                     int confirm = JOptionPane.showOptionDialog(DataPartDialog.this,
706                         "Invalid type.\nDo you want to discard the list operation?
707                         ",
708                         "Invalid component", JOptionPane.YES_NO_OPTION,
709                         JOptionPane.QUESTION_MESSAGE, null, null, null);
710                     if (confirm == 0) {
711                         DataPartDialog.this.dispose();
712                     }
713                 }
714             };
715         }
716         dialogPanel = new DialogPanel() {
717             @Override
718             public void applyOperationExit() {
719                 lang.type.Type t = opTypePanel.getType();
720                 if (t.isConcrete(true)) {
721                     dataFlowPanel.addNewDataPartListOperation(DataPartListOperation.
722                         ListOperationType.PREPEND, Arrays.asList(t), Arrays.asList(t));
723                     DataPartDialog.this.dispose();
724                 } else {
725                     int confirm = JOptionPane.showOptionDialog(DataPartDialog.this,
726                         "Invalid type.\nDo you want to discard the list operation?
727                         ",
728                         "Invalid component", JOptionPane.YES_NO_OPTION,
729                         JOptionPane.QUESTION_MESSAGE, null, null, null);
730                     if (confirm == 0) {
```

```
728                     DataPartDialog.this.dispose();
729                 }
730             }
731         }
732     };
733 } else if (cbox.getSelectedItem().equals("Augment")) {
734     panel.add(opSplitPanel);
735     dialogPanel = new DialogPanel() {
736         @Override
737         public void applyOperationExit() {
738             lang.type.Type t1 = opTypePanel1.getType();
739             lang.type.Type t2 = opTypePanel2.getType();
740             if (t1.isConcrete(true) && t2.isConcrete(true)) {
741                 dataFlowPanel.addNewDataPartListOperation(DataPartListOperation.
742                     ListOperationType.AUGMENT, Arrays.asList(t1, t2), Arrays.asList(t1, t2));
743                 DataPartDialog.this.dispose();
744             } else {
745                 int confirm = JOptionPane.showOptionDialog(DataPartDialog.this,
746                     "Invalid type.\nDo you want to discard the list operation?
747                     ",
748                     "Invalid component", JOptionPane.YES_NO_OPTION,
749                     JOptionPane.QUESTION_MESSAGE, null, null, null);
750                 if (confirm == 0) {
751                     DataPartDialog.this.dispose();
752                 }
753             }
754         } else if (cbox.getSelectedItem().equals("Length")) {
755             dialogPanel = new DialogPanel() {
756                 @Override
757                 public void applyOperationExit() {
758                     dataFlowPanel.addNewDataPartListOperation(DataPartListOperation.
759                     ListOperationType.LENGTH);
760                     DataPartDialog.this.dispose();
761                 }
762             };
763         } else {
764             throw new RuntimeException("unexpected cbox item!");
765         }
766         panel.revalidate();
767         panel.repaint();
768     });
769 }
770 return panel;
771 }
772
773 private void setServicePanel() {
774 }
775
776 private void setFunctionalPanel() {
777     JComboBox<String> outerComboBox = new JComboBox<>(functionalComboBoxOptions);
778     outerComboBox.setSelectedItem(null);
779     outerComboBox.setFont(UtilFont.textFont);
```

```
780
781     JLabel outerComboBoxLabel = new JLabel("Functional type");
782     outerComboBoxLabel.setFont(UtilFont.textFont);
783
784     JPanel outerComobBoxPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));
785     outerComobBoxPanel.add(outerComboBoxLabel);
786     outerComobBoxPanel.add(outerComboBox);
787     //outerComobBoxPanel.setBorder(BorderFactory.createMatteBorder(1, 0, 1, 0, Color.black));
788
789     JPanel outerPanel = new JPanel(new BorderLayout());
790     outerPanel.add(outerComobBoxPanel, BorderLayout.NORTH);
791
792     outerComboBox.addItemListener(new ItemListener() {
793         @Override
794         public void itemStateChanged(ItemEvent e) {
795             dialogPanel = new DialogPanel() {
796                 @Override
797                 public void applyOperationExit() {
798                     int confirm = JOptionPane.showOptionDialog(DataPartDialog.this,
799                             "Nothing selected.\nDo you want to close?", "Close confirmation", JOptionPane.YES_NO_OPTION,
800                             JOptionPane.QUESTION_MESSAGE, null, null, null);
801                     if (confirm == 0) {
802                         DataPartDialog.this.dispose();
803                     }
804                 }
805             };
806         };
807
808         if (currentFunctionalPanel != null)
809             outerPanel.remove(currentFunctionalPanel);
810         switch (outerComboBox.getSelectedIndex()) {
811             case FUN_ARITHMETIC_INDEX:
812                 currentFunctionalPanel = getFunctionalArithmeticPanel();
813                 break;
814             case FUN_BOOLEAN_INDEX:
815                 currentFunctionalPanel = getFunctionalBooleanPanel();
816                 break;
817             case FUN_STRING_OPERATION_INDEX:
818                 currentFunctionalPanel = getFunctionalStringPanel();
819                 break;
820             case FUN_COMPARISON_INDEX:
821                 currentFunctionalPanel = getFunctionalComparisonPanel();
822                 break;
823             case FUN_LIST_OPERATION_INDEX:
824                 currentFunctionalPanel = getFunctionalListPanel();
825                 break;
826             default:
827                 throw new RuntimeException("index unexpected");
828         }
829         outerPanel.add(currentFunctionalPanel);
830         outerPanel.revalidate();
831         outerPanel.repaint();
832     });
833 });
834
```

```
835     dialogPanel = new DialogPanel() {
836         @Override
837         public void applyOperationExit() {
838             int confirm = JOptionPane.showOptionDialog(DataPartDialog.this,
839                 "Nothing selected.\nDo you want to close?",
840                 "Close confirmation", JOptionPane.YES_NO_OPTION,
841                 JOptionPane.QUESTION_MESSAGE, null, null, null);
842             if (confirm == 0) {
843                 DataPartDialog.this.dispose();
844             }
845         }
846     };
847
848     resetScrollPane(outerPanel);
849 }
850
851     public void setTypeCastPanel() {
852         JPanel panel = new JPanel(new BorderLayout());
853         TypeSelector leftSel = new TypeSelector("Cast from", true);
854         TypeSelector rightSel = new TypeSelector("Cast to");
855         JSplitPane splitPane = new JSplitPane(
856             JSplitPane.HORIZONTAL_SPLIT, new UtilScrollPane(leftSel),
857             new UtilScrollPane(rightSel));
858         splitPane.setDividerLocation(0.5);
859         splitPane.setResizeWeight(0.5);
860         panel.add(splitPane);
861
862         dialogPanel = new DialogPanel() {
863             @Override
864             public void applyOperationExit() {
865                 try {
866                     dataFlowPanel.addNewDataPartCast(leftSel.getType(), leftSel.getType(),
867                         rightSel.getType());
868                     DataPartDialog.this.dispose();
869                 } catch (InvalidDataPartException e) {
870                     int confirm = JOptionPane.showOptionDialog(DataPartDialog.this,
871                         e.getMessage() + ".\nDo you want to discard the type cast?",
872                         "Invalid component", JOptionPane.YES_NO_OPTION,
873                         JOptionPane.QUESTION_MESSAGE, null, null, null);
874                     if (confirm == 0) {
875                         DataPartDialog.this.dispose();
876                     }
877                 }
878             };
879
880             resetScrollPane(panel);
881         }
882
883     private void setSubdataflowPanel() {
884         JPanel panel = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));
885         JTextField titleField = new JTextField();
886         titleField.setPreferredSize(new Dimension(200, titleField.getPreferredSize().height));
887         UtilPrompt.setPrompt("Subdataflow title ", titleField);
888         panel.add(titleField);
```

```
889
890     dialogPanel = new DialogPanel() {
891         @Override
892         public void applyOperationExit() {
893             try {
894                 dataFlowPanel.addNewDataPartSubFlow(titleField.getText());
895                 DataPartDialog.this.dispose();
896             } catch (TabbedPane.InvalidTitleException | InvalidDataPartException e) {
897                 int confirm = JOptionPane.showOptionDialog(DataPartDialog.this,
898                     e.getMessage() + ".\nDo you want to discard the subdataflow?",
899                     "Close confirmation", JOptionPane.YES_NO_OPTION,
900                     JOptionPane.QUESTION_MESSAGE, null, null, null);
901                 if (confirm == 0) {
902                     DataPartDialog.this.dispose();
903                 }
904             }
905         }
906     };
907
908     resetScrollPane(panel);
909 }
910
911     private void setConstantPanel(Graphics g) {
912         ConstantDefiner cd = new ConstantDefiner(g);
913         dialogPanel = new DialogPanel() {
914             @Override
915             public void applyOperationExit() {
916                 try {
917                     Constant val = cd.getValue();
918                     dataFlowPanel.addNewDataPartConstant(val);
919                     DataPartDialog.this.dispose();
920                 } catch (InvalidConstantException e) {
921                     int confirm = JOptionPane.showOptionDialog(DataPartDialog.this,
922                         e.getMessage() + ".\nDo you want to discard the constant?",
923                         "Close confirmation", JOptionPane.YES_NO_OPTION,
924                         JOptionPane.QUESTION_MESSAGE, null, null, null);
925                     if (confirm == 0) {
926                         DataPartDialog.this.dispose();
927                     }
928                 }
929             }
930         };
931         resetScrollPane(cd);
932     }
933
934     private void setSinkPanel() {
935         dialogPanel = new DialogPanel() {
936             @Override
937             public void applyOperationExit() {
938                 dataFlowPanel.addNewDataPartSink(null);
939                 DataPartDialog.this.dispose();
940             }
941         };
942         resetScrollPane(new JPanel());
943     }
```

```

944
945     private void setTypePanel(String typeName, BiPredicate<String, lang.type.Type> cons, boolean
946         allowAuto) {
947         JPanel outer = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));
948         JPanel panel = new JPanel();
949         panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
950         JTextField inputField = new JTextField();
951         inputField.setFont(UtilFont.textField);
952         UtilPrompt.setPrompt("Enter "+typeName+" name", inputField);
953         inputField.setPreferredSize(new Dimension(200, inputField.getPreferredSize().height));
954         JPanel inputPanel = new JPanel(new GridBagLayout());
955         inputPanel.add(inputField);
956         panel.add(inputPanel);
957         panel.add(Box.createRigidArea(new Dimension(0,5)));
958         TypeSelector ts = new TypeSelector("Type", allowAuto);
959         panel.add(ts);
960         dialogPanel = new DialogPanel() {
961             @Override
962             public void applyOperationExit() {
963                 lang.type.Type type = ts.getType();
964                 if (cons.test(inputField.getText(), type))
965                     DataPartDialog.this.dispose();
966             }
967         };
968         outer.add(panel);
969         resetScrollPane(outer);
970     }
971
972     private void setParameterPanel() {
973         setTypePanel("input", (n,t) -> {
974             try {
975                 if (!t.isConcrete(true))
976                     throw new InvalidDataPartException("Invalid type");
977                 dataFlowPanel.addDataPartParameter(n, t, t, dataFlowPanel.getParentTabbedPane()
978                     .getChildParamaterDeleteEvent());
979                 return true;
980             } catch (InvalidDataPartException e) {
981                 int confirm = JOptionPane.showOptionDialog(DataPartDialog.this,
982                     e.getMessage()+"\nDo you want to discard the input?",
983                     "Close confirmation", JOptionPane.YES_NO_OPTION,
984                     JOptionPane.QUESTION_MESSAGE, null, null, null);
985                 if (confirm == 0) {
986                     DataPartDialog.this.dispose();
987                 }
988                 return false;
989             }
990         }, DataPartParameter.allowAuto(dataFlowPanel));
991     }
992
993     private void setResultPanel() {
994         setTypePanel("output", (n,t) -> {
995             try {
996                 if (!t.isConcrete(true))
997                     throw new InvalidDataPartException("Invalid type");

```

```
996         dataFlowPanel.addNewDataPartResult(n, t, t, dataFlowPanel.getParentTabbedPane()).
997         getChildResultDeleteEvent());
998     }
999     } catch (InvalidDataPartException e) {
1000         int confirm = JOptionPane.showOptionDialog(DataPartDialog.this,
1001             e.getMessage()+"\nDo you want to discard the output?",
1002             "Close confirmation", JOptionPane.YES_NO_OPTION,
1003             JOptionPane.QUESTION_MESSAGE, null, null, null);
1004         if (confirm == 0) {
1005             DataPartDialog.this.dispose();
1006         }
1007     }
1008 }, true);
1009 }
1010
1011 private void setRecordConstructorPanel() {
1012     TypeSelector typePanel = new TypeSelector("Record type");
1013     JPanel panel = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));
1014     panel.add(typePanel);
1015     dialogPanel = new DialogPanel() {
1016         @Override
1017         public void applyOperationExit() {
1018             lang.type.Type t = typePanel.getType();
1019             if (!t.isConcrete(true)) {
1020                 int confirm = JOptionPane.showOptionDialog(DataPartDialog.this,
1021                     "Invalid type.\nDo you want to discard the record constructor?",
1022                     "Close confirmation", JOptionPane.YES_NO_OPTION,
1023                     JOptionPane.QUESTION_MESSAGE, null, null, null);
1024                 if (confirm == 0) {
1025                     DataPartDialog.this.dispose();
1026                 }
1027             } else if (t.getTypeCode() != TypeCode.RECORD) {
1028                 int confirm = JOptionPane.showOptionDialog(DataPartDialog.this,
1029                     "Type must be record.\nDo you want to discard the record constructor?"
1030                     ,
1031                     "Close confirmation", JOptionPane.YES_NO_OPTION,
1032                     JOptionPane.QUESTION_MESSAGE, null, null, null);
1033                 if (confirm == 0) {
1034                     DataPartDialog.this.dispose();
1035                 }
1036             } else {
1037                 dataFlowPanel.addNewDataPartRecordConstructor((TypeRecord)t);
1038                 DataPartDialog.this.dispose();
1039             }
1040         };
1041         resetScrollPane(panel);
1042     }
1043
1044     private static interface DialogPanel {
1045         public void applyOperationExit();
1046     }
1047 }
```

LISTING E.87: gui/dataflow/DataPartDialog.java

```
1 package gui.dataflow;
2
3 import java.awt.Graphics;
4 import java.awt.Window;
5
6 import gui.DataFlowPanel;
7 import gui.MainWindow;
8 import lang.type.TypeBool;
9 import lang.type.TypeNumber;
10 import lang.type.TypeString;
11
12 public class DataPartStringOperation extends DataPart {
13     public enum StringOperationType {LENGTH, CONCATENATE, CONTAINS}
14
15     private final static Port[] containsInputs = {new Port("String", new TString()),
16         new Port("Substring", new TString())};
17     private final static Port[] binaryInputs =
18         {new Port("!0", new TString()), new Port("!1", new TString())};
19     private final static Port[] unaryInput = {new Port("!0", new TString())};
20     private final static Port[] stringResult = {new Port("!0", new TString())};
21     private final static Port[] boolResult = {new Port("!0", new TypeBool())};
22     private final static Port[] numberResult = {new Port("!0", new TypeNumber())};
23
24     private static Port[] getInputPorts(StringOperationType type) {
25         switch (type) {
26             case LENGTH:
27                 return unaryInput;
28             case CONTAINS:
29                 return containsInputs;
30             default:
31                 return binaryInputs;
32         }
33     }
34
35     private static String getStringOperationTitle(StringOperationType type) {
36         switch (type) {
37             case LENGTH:
38                 return "Length";
39             case CONCATENATE:
40                 return "Concatenate";
41             case CONTAINS:
42                 return "Contains";
43             default:
44                 throw new RuntimeException("unexpected string operation type " + type);
45         }
46     }
47
48     private static Port[] getOutputPorts(StringOperationType type) {
49         switch (type) {
50             case CONTAINS:
51                 return boolResult;
```

```

52     case LENGTH:
53         return numberResult;
54     default:
55         return stringResult;
56     }
57 }
58
59 private StringOperationType stringOperationType;
60
61 public DataPartStringOperation(StringOperationType type, DataFlowPanel dfp, Graphics g) {
62     super(getInputPorts(type), getOutputPorts(type), getStringOperationTitle(type), "String",
63           dfp, g);
64     stringOperationType = type;
65 }
66
67 public StringOperationType getStringOperationType() {
68     return stringOperationType;
69 }
70
71 @Override
72 public boolean canFail() {
73     return false;
74 }
75
76 @Override
77 public void doubleClick(MainWindow mainWindow, Window parent) {}
78
79 @Override
80 public boolean markDeletedIfAllowed() {
81     return true;
82 }

```

LISTING E.88: gui/dataflow/DataPartStringOperation.java

```

1 package gui.dataflow;
2
3 import java.awt.Graphics;
4 import java.awt.Window;
5 import java.util.List;
6
7 import gui.DataFlowPanel;
8 import gui.MainWindow;
9 import lang.type.RecordColumn;
10 import lang.type.TypeRecord;
11
12 public class DataPartRecordConstructor extends DataPart {
13
14     private static Port[] getInputPorts(TypeRecord type) {
15         List<RecordColumn> cols = type.getColumnsInOrder();
16         Port[] ret = new Port[cols.size()];
17         for (int i = 0; i < ret.length; i++) {
18             RecordColumn c = cols.get(i);
19             ret[i] = new Port(c.getId(), c.getType());
20         }

```

```

21         return ret;
22     }
23
24     private static Port[] getOutputPort(TypeRecord type) {
25         return new Port[]{new Port("!0", type)};
26     }
27
28     private TypeRecord recordType;
29
30     public DataPartRecordConstructor(TypeRecord type, DataFlowPanel dfp, Graphics g) {
31         super(getInputPorts(type), getOutputPort(type), "Record", "Constructor", dfp, g);
32         recordType = type;
33     }
34
35     public TypeRecord getRecordType() {
36         return recordType;
37     }
38
39     @Override
40     public boolean canFail() {
41         return false;
42     }
43
44     @Override
45     public void doubleClick(MainWindow mainWindow, Window parent) {}
46
47     @Override
48     public boolean markDeletedIfAllowed() {
49         return true;
50     }
51 }
```

LISTING E.89: gui/dataflow/DataPartRecordConstructor.java

```

1 package gui.dataflow;
2
3 import java.awt.BorderLayout;
4 import java.awt.Color;
5 import java.awt.Dimension;
6 import java.awt.FlowLayout;
7 import java.awt.Graphics;
8 import java.awt.GridBagLayout;
9 import java.awt.Window;
10 import java.awt.event.ActionEvent;
11 import java.awt.event.ActionListener;
12 import java.awt.event.WindowAdapter;
13 import java.awt.event.WindowEvent;
14 import java.util.function.Predicate;
15
16 import javax.swing.JButton;
17 import javax.swing.JDialog;
18 import javax.swing.JOptionPane;
19 import javax.swing.JPanel;
20 import javax.swing.JTextField;
21
```



```
73     void setType(Type t) {
74         type = t;
75     }
76
77     public Type getType() {
78         return type;
79     }
80
81     @Override
82     public boolean canFail() {
83         return false;
84     }
85
86     public void changeType(Type newType) {
87         changeOutputPort(new Port("", type), new Port("", newType));
88         disconnectOutputTypeIfConnected("");
89         type = newType;
90         getDataFlowPanel().getParentTabbedPane().parameterTypeChanged(getDataFlowPanel(), this);
91     }
92
93     public void changeName(String newName) {
94         String oldName = parameterName;
95         parameterName = newName;
96         setTitle(parameterName);
97         getDataFlowPanel().getParentTabbedPane().changeParameterName(
98             getDataFlowPanel(), DataPartParameter.this, oldName);
99     }
100
101    @Override
102    public void doubleClick(MainWindow mainWindow, Window parent) {
103        ShowTypePanel showTypePanel = new ShowTypePanel(type, true);
104        showTypePanel.setLayout(new GridBagLayout());
105        showTypePanel.setBackground(Color.white);
106        JDialog dialog = new JDialog(parent);
107        dialog.setSize(new Dimension(500, 300));
108        dialog.setLocationRelativeTo(parent);
109        dialog.setModal(true);
110        dialog.getContentPane().add(showTypePanel);
111
112        JButton changeNameButton = new UtilButton("Change name");
113        JButton changeTypeButton = new UtilButton("Change type");
114        JPanel changePanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));
115        changePanel.add(changeNameButton);
116        changePanel.add(changeTypeButton);
117
118        JPanel namePanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));
119        namePanel.add(new UtilLabel(getName()));
120
121        changeNameButton.addActionListener(new ActionListener() {
122            @Override
123            public void actionPerformed(ActionEvent e) {
124                dialog.getContentPane().remove(changePanel);
125                dialog.getContentPane().remove(showTypePanel);
126                JPanel p = new JPanel(new GridBagLayout());
127                JTextField field = new JTextField();
```

```

128         UtilPrompt.setPrompt("New name", field);
129         field.setPreferredSize(new Dimension(200, field.getPreferredSize().height));
130         p.add(field);
131         dialog.add(p);
132         dialog.revalidate();
133         dialog.repaint();
134         dialog.setDefaultCloseOperation(JDialog.DO_NOTHING_ON_CLOSE);
135         dialog.addWindowListener(new WindowAdapter() {
136             @Override
137             public void windowClosing(WindowEvent e) {
138                 String newName = field.getText();
139                 if (!UtilRegex.isIdentifier(newName)) {
140                     int confirm = JOptionPane.showOptionDialog(dialog,
141                         "Invalid input name '"+newName+"'.\nDo you want to discard the
142                         change?",
143                         "Close confirmation", JOptionPane.YES_NO_OPTION,
144                         JOptionPane.QUESTION_MESSAGE, null, null, null);
145                     if (confirm == 0) {
146                         dialog.dispose();
147                         return;
148                     }
149                 }
150                 try {
151                     getDataFlowPanel().validateAddDataPartParameter(newName);
152                 } catch (InvalidDataPartException e1) {
153                     int confirm = JOptionPane.showOptionDialog(dialog,
154                         e1.getMessage()+"\nDo you want to discard the change?",
155                         "Close confirmation", JOptionPane.YES_NO_OPTION,
156                         JOptionPane.QUESTION_MESSAGE, null, null, null);
157                     if (confirm == 0) {
158                         dialog.dispose();
159                         return;
160                     }
161                 }
162                 changeName(newName);
163                 dialog.dispose();
164             }
165         });
166     });
167 }
168 });
169
170 changeTypeButton.addActionListener(new ActionListener() {
171     @Override
172     public void actionPerformed(ActionEvent e) {
173         if (!allowChangeType) {
174             JOptionPane.showMessageDialog(dialog, "Cannot change type of this input",
175                         "Unable to change type", JOptionPane.ERROR_MESSAGE);
176             return;
177         }
178         if (getDataFlowPanel().getParentTabbedPane().isParameterConnected(
179             getDataFlowPanel(), DataPartParameter.this)) {
String tabbedType = getDataFlowPanel().getParentTabbedPane() instanceof
DataFlowPanel ?

```

```

180                     "dataflow" : "controlflow";
181             int confirm = JOptionPane.showOptionDialog(dialog,
182                         "Change will affect "+tabbedType+" "+
183                         getDataFlowPanel().getParentTabbedPane().getTitle()+"?.\nDo
184                         you want to change?", 
185                         "Change type confirmation", JOptionPane.YES_NO_OPTION,
186                         JOptionPane.QUESTION_MESSAGE, null, null, null);
187             if (confirm != 0)
188                 return;
189             dialog.getContentPane().remove(changePanel);
190             dialog.getContentPane().remove(showTypePanel);
191             TypeSelector sel = new TypeSelector("New type", allowAuto(getDataFlowPanel()));
192             dialog.add(new UtilScrollPane(sel));
193             dialog.revalidate();
194             dialog.repaint();
195             dialog.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
196             dialog.addWindowListener(new WindowAdapter() {
197                 @Override
198                 public void windowClosing(WindowEvent e) {
199                     Type t = sel.getType();
200                     /*if (t.equals(type)) {
201                         dialog.dispose();
202                         return;
203                     }*/
204                     if (!t.isConcrete(true)) {
205                         int confirm = JOptionPane.showOptionDialog(dialog,
206                                         "Invalid type.\nDo you want to discard the change?", 
207                                         "Close confirmation", JOptionPane.YES_NO_OPTION,
208                                         JOptionPane.QUESTION_MESSAGE, null, null, null);
209                         if (confirm == 0) {
210                             dialog.dispose();
211                             return;
212                         } else {
213                             return;
214                         }
215                     }
216                     changeType(t);
217                     dialog.dispose();
218                 }
219             });
220         });
221     });
222     dialog.getContentPane().add(namePanel, BorderLayout.NORTH);
223     dialog.getContentPane().add(changePanel, BorderLayout.SOUTH);
224     dialog.setVisible(true);
225 }
226
227 @Override
228 public boolean markDeletedIfAllowed() {
229     boolean b = deleteEvent.test(this);
230     return b;
231 }
232 }
```

LISTING E.90: gui/dataflow/DataPartParameter.java

```
1 package gui.dataflow;
2
3 import java.awt.Graphics;
4 import java.awt.Window;
5 import java.util.Arrays;
6
7 import gui.DataFlowPanel;
8 import gui.MainWindow;
9 import lang.type.Type;
10 import lang.type.TypeAuto;
11 import lang.type.TypeAutoChangeException;
12 import lang.type.TypeAutoInputListener;
13
14 public class DataPartSink extends DataPart implements TypeAutoInputListener {
15     private static final Port[] inputPort = {new Port("", new TypeAuto())};
16     private static final Port[] output = {};
17     private Type type;
18
19     private static Port[] getInputPort(Type t) {
20         return t == null ? inputPort : new Port[]{new Port("", t)};
21     }
22
23     public DataPartSink(Type type, DataFlowPanel dfp, Graphics g) {
24         super(getInputPort(type), output, "", "Sink", dfp, g);
25         if (type == null) {
26             //System.out.println("sink default type "+inputPort[0].getType());
27             this.type = inputPort[0].getType();
28         } else {
29             //System.out.println("sink type "+type);
30             this.type = type;
31         }
32         setInputArrowListeners(Arrays.asList(this));
33     }
34
35     public Type getType() {
36         return type;
37     }
38
39     @Override
40     public boolean canFail() {
41         return false;
42     }
43
44     @Override
45     public void doubleClick(MainWindow mainWindow, Window parent) {}
46
47     @Override
48     public boolean markDeletedIfAllowed() {
49         return true;
50     }
51
```

```

52     @Override
53     public void autoTypeInputChanged(Type newType) throws TypeAutoChangeException {
54         type = newType;
55         //System.out.println("sink change input type "+newType);
56         changeInputPortType("", newType);
57     }
58
59     @Override
60     public void autoTypeInputRemoved() {
61         type = inputPort[0].getType();
62         //System.out.println("sink reset input type "+type);
63         changeInputPortType("", inputPort[0].getType());
64     }
65
66     @Override
67     public boolean autoTypeInputMayChange() {
68         return true;
69     }
70 }
```

LISTING E.91: gui/dataflow/DataPartSink.java

```

1 package gui.dataflow;
2
3 import java.awt.Graphics;
4 import java.awt.Window;
5 import java.util.Arrays;
6
7 import util.UtilRegex;
8 import gui.DataFlowPanel;
9 import gui.MainWindow;
10 import gui.DataFlowPanel.EventReceiver;
11 import gui.TabbedPane.InvalidTitleException;
12 import gui.TabbedPane.TitleChangedListener;
13 import lang.type.Type;
14 import lang.type.TypeAutoChangeException;
15 import lang.type.TypeAutoInputListener;
16 import lang.type.TypeAutoOutputListener;
17 import lang.type.TypeUnknown;
18
19 public class DataPartSubFlow extends DataPart implements TitleChangedListener, EventReceiver,
20     IDataPartSubFlow {
21     private DataFlowPanel subDataFlowPanel;
22     private MainWindow mainWindow;
23
24     private class InputListener implements TypeAutoInputListener, TypeAutoOutputListener {
25         private DataPartParameter input;
26
27         private InputListener(DataPartParameter input) {
28             this.input = input;
29         }
30
31         @Override
32         public void autoTypeInputChanged(Type newType) throws TypeAutoChangeException {
33             changeInputPortType(input.getName(), newType);
```

```
33         this.input.setType(newType);
34         this.input.changeOutputPortType("", newType);
35     }
36
37     @Override
38     public void autoTypeInputRemoved() {
39         if (autoTypeInputMayChange()) {
40             changeInputPortType(input.getName(), input.getOriginalType());
41             this.input.setType(input.getOriginalType());
42             this.input.changeOutputPortType("", input.getOriginalType());
43         }
44     }
45
46     @Override
47     public boolean autoTypeInputMayChange() {
48         return !this.input.isOutputPortConnected(new Port("", new TypeUnknown(""))));
49     }
50
51     @Override
52     public void autoTypeOutputRemoved() {
53         if (!isInputPortConnected(new Port(input.getName(), new TypeUnknown(""))))
54             autoTypeInputRemoved();
55     }
56 }
57
58     public DataPartSubFlow(String dpTitle, DataFlowPanel parentFlow,
59                           MainWindow mainWin, Graphics g) throws InvalidDataPartException, InvalidTitleException
60     {
61         super(new Port[] {}, new Port[] {}, dpTitle, "Dataflow", parentFlow, g);
62         this.subDataFlowPanel = new DataFlowPanel(mainWin, dpTitle, parentFlow, this);
63         mainWin.addExplorerItem(parentFlow, subDataFlowPanel);
64         subDataFlowPanel.addTitleChangedListener(this);
65         mainWindow = mainWin;
66         if(!UtilRegex.isIdentifier(dpTitle))
67             throw new InvalidDataPartException("Invalid dataflow name '"+dpTitle+"'");
68     }
69
70     @Override
71     public boolean canFail() {
72         return subDataFlowPanel.canFail();
73     }
74
75     @Override
76     public void doubleClick(MainWindow mainWindow, Window parent) {
77         mainWindow.openExplorerItem(subDataFlowPanel);
78     }
79
80     @Override
81     public boolean markDeletedIfAllowed() {
82         subDataFlowPanel.removeTitleChangedListener(this);
83         mainWindow.removeExplorerItem(subDataFlowPanel);
84         return true;
85     }
86
87     @Override
```

```

87     public boolean addParameterIfAllowed(DataPartParameter param) {
88         addInputPort(new Port(param.getName(), param.getType()));
89         InputListener listener = new InputListener(param);
90         setInputArrowListener(param.getName(), listener);
91         param.setOutputArrowListener("", listener);
92         return true;
93     }
94
95     @Override
96     public boolean addResultIfAllowed(DataPartResult result) {
97         addOutputPort(new Port(result.getName(), result.getType()));
98         setOutputArrowListener(result.getName(), result);
99         result.setInputArrowListeners(Arrays.asList(result));
100        return true;
101    }
102
103    @Override
104    public boolean removeParameterIfAllowed(DataPartParameter param) {
105        removeInputPort(new Port(param.getName(), param.getType()));
106        return true;
107    }
108
109    @Override
110    public boolean removeResultIfAllowed(DataPartResult result) {
111        removeOutputPort(new Port(result.getName(), result.getType()));
112        return true;
113    }
114
115    @Override
116    public void titleChanged(String title) {
117        setTitle(title);
118    }
119
120    @Override
121    public DataFlowPanel getSubDataFlowPanel() {
122        return subDataFlowPanel;
123    }
124 }
```

LISTING E.92: gui/dataflow/DataPartSubFlow.java

```

1 package gui.dataflow;
2
3 import java.awt.Graphics;
4 import java.awt.Point;
5 import java.awt.Window;
6 import java.util.ArrayList;
7 import java.util.Arrays;
8 import java.util.HashMap;
9 import java.util.List;
10 import java.util.Map;
11
12 import javax.swing.JOptionPane;
13
14 import lang.type.RecordColumn;
```

```
15 import lang.type.Type;
16 import lang.type.TypeAuto;
17 import lang.type.TypeAutoChangeException;
18 import lang.type.TypeAutoInputListener;
19 import lang.type.TypeAutoOutputListener;
20 import lang.type.TypeBool;
21 import lang.type.TypeList;
22 import lang.type.TypeRecord;
23 import lang.type.TypeUnknown;
24 import gui.DataFlowPanel;
25 import gui.MainWindow;
26 import gui.DataFlowPanel.EventReceiver;
27 import gui.TabbedPanel.InvalidTitleException;
28
29 public class DataPartFunctional extends DataPart implements EventReceiver, IDataPartSubFlow {
30     private final static TypeBool typeBool = new TypeBool();
31     private final static TypeAuto typeAuto = new TypeAuto();
32     private final static Port[] autoPort = {new Port("", typeAuto)};
33     private final static Port[] accumInputPorts =
34         {new Port("List", new TypeList(typeAuto)), new Port("Accum", typeAuto)};
35     private final static Port[] autoListPort = {new Port("", new TypeList(typeAuto))};
36     private final static Port[] joinInputPorts = {
37         new Port("Left", new TypeList(typeAuto)), new Port("Right", new TypeList(typeAuto))};
38     private final static Port[] voidPort = {};
39
40     public enum FunctionalType {MAP, FILTER, JOIN, FOREACH, ACCUMULATE};
41
42     private DataFlowPanel subDataFlowPanel;
43     private MainWindow mainWindow;
44     private int numberMissingInputs;
45     private int numberMissingOutputs;
46     private IFunctional functionalImpl;
47     private FunctionalType functionalType;
48
49     private ArrayList<DataPartParameter> subInputs = new ArrayList<>();
50     private ArrayList<DataPartResult> subOutputs = new ArrayList<>();
51
52     private static String getFunctionalTitle(FunctionalType type) {
53         switch (type) {
54             case MAP:
55                 return "Map";
56             case FILTER:
57                 return "Filter";
58             case JOIN:
59                 return "Join";
60             case FOREACH:
61                 return "For_each";
62             case ACCUMULATE:
63                 return "Accumulate";
64             default:
65                 throw new IllegalArgumentException("unexpected type " + type);
66         }
67     }
68
69     private static Port[] getDefaultInputPorts(FunctionalType type) {
```

```
70     switch (type) {
71     case MAP:
72         return autoListPort;
73     case FILTER:
74         return autoListPort;
75     case JOIN:
76         return joinInputPorts;
77     case FOREACH:
78         return autoListPort;
79     case ACCUMULATE:
80         return accumInputPorts;
81     default:
82         throw new IllegalArgumentException("unexpected type " + type);
83     }
84 }
85
86 private static Port[] getDefaultOutputPorts(FunctionalType type) {
87     switch (type) {
88     case MAP:
89         return autoListPort;
90     case FILTER:
91         return autoListPort;
92     case JOIN:
93         ArrayList<RecordColumn> cols = new ArrayList<>();
94         cols.add(new RecordColumn("Left", typeAuto));
95         cols.add(new RecordColumn("Right", typeAuto));
96         return new Port[]{new Port("", new TypeRecord(cols))};
97     case FOREACH:
98         return voidPort;
99     case ACCUMULATE:
100        return autoPort;
101    default:
102        throw new IllegalArgumentException("unexpected type " + type);
103    }
104 }
105
106 public DataPartFunctional(FunctionalType type, MainWindow mainWin, String subDataFlowTitle,
107                           DataFlowPanel dfp, Graphics g) throws InvalidTitleException, InvalidDataPartException
108 {
109     this(type, mainWin, subDataFlowTitle, dfp, g, null, null);
110 }
111
112 public DataPartFunctional(FunctionalType type, MainWindow mainWin, String subDataFlowTitle,
113                           DataFlowPanel dfp, Graphics g, List<RecordColumn> subInputs, List<RecordColumn>
114                           subOutputs)
115                           throws InvalidTitleException, InvalidDataPartException {
116     super(getDefaultInputPorts(type), getDefaultOutputPorts(type),
117           getFunctionalTitle(type), DataPartListOperation.subtitle, dfp, g);
118     this.subDataFlowPanel = new DataFlowPanel(mainWin, subDataFlowTitle, dfp, this);
119     mainWin.addExplorerItem(dfp, subDataFlowPanel);
120     mainWindow = mainWin;
121     functionalType = type;
122     if (subInputs != null) {
123         dfp.addTempComponent(this);
124     }
125 }
```

```
123         addDataFlowPanelIO(type, subInputs, subOutputs);
124     }
125
126     private void addMap(List<RecordColumn> subInputs, List<RecordColumn> subOutputs) throws
127         InvalidDataPartException {
128         numberMissingInputs = 1;
129         numberMissingOutputs = 1;
130         DataPartParameter p = subDataFlowPanel.addNewDataPartParameter("Item", typeAuto, typeAuto,
131             (x) -> true, true);
132         this.subInputs.add(p);
133         p.setPointBoxLocation(new Point(50, 40));
134         DataPartResult r = subDataFlowPanel.addNewDataPartResult("Item", typeAuto, typeAuto, (x)
135             -> true, true);
136         this.subOutputs.add(r);
137         r.setPointBoxLocation(new Point(50, 200));
138         functionalImpl = new FunctionalMap();
139
140         setOutputArrowListener("", r);
141         r.setInputArrowListeners(Arrays.asList(r));
142
143         if (subInputs != null) {
144             RecordColumn in = subInputs.get(0);
145             RecordColumn out = subOutputs.get(0);
146             p.changeType(in.getType());
147             p.changeName(in.getId());
148             r.changeType(out.getType());
149             r.changeName(out.getId());
150         }
151     }
152
153     private void addFilter(List<RecordColumn> subInputs, List<RecordColumn> subOutputs) throws
154         InvalidDataPartException {
155         numberMissingInputs = 1;
156         numberMissingOutputs = 1;
157         DataPartParameter p = subDataFlowPanel.addNewDataPartParameter("Item", typeAuto, typeAuto,
158             (x) -> true, true);
159         this.subInputs.add(p);
160         p.setPointBoxLocation(new Point(50, 40));
161         DataPartResult r = subDataFlowPanel.addNewDataPartResult("Keep", typeBool, typeBool, (x)
162             -> true, false);
163         this.subOutputs.add(r);
164         r.setPointBoxLocation(new Point(50, 200));
165         functionalImpl = new FunctionalFilter();
166
167         if (subInputs != null) {
168             assert subOutputs != null;
169             RecordColumn in = subInputs.get(0);
170             RecordColumn out = subOutputs.get(0);
```

```
171     private void addJoin(List<RecordColumn> subInputs, List<RecordColumn> subOutputs) throws
172         InvalidDataPartException {
173             numberMissingInputs = 2;
174             numberMissingOutputs = 1;
175             DataPartParameter p1 = subDataFlowPanel.addNewDataPartParameter("Left_item", typeAuto,
176                 typeAuto, (x) -> true, true);
177             p1.setPointBoxLocation(new Point(50, 40));
178             DataPartParameter p2 = subDataFlowPanel.addNewDataPartParameter("Right_item", typeAuto,
179                 typeAuto, (x) -> true, true);
180             p2.setPointBoxLocation(new Point(150, 40));
181             this.subInputs.add(p1);
182             this.subInputs.add(p2);
183             DataPartResult r = subDataFlowPanel.addNewDataPartResult("Keep", typeBool, typeBool, (x)
184                 -> true, false);
185             this.subOutputs.add(r);
186             r.setPointBoxLocation(new Point(100, 200));
187             functionalImpl = new FunctionalJoin();
188
189
190         if (subInputs != null) {
191             RecordColumn in = subInputs.get(0);
192             p1.changeType(in.getType());
193             p1.changeName(in.getId());
194             in = subInputs.get(1);
195             p2.changeType(in.getType());
196             p2.changeName(in.getId());
197             RecordColumn out = subOutputs.get(0);
198             r.changeName(out.getId());
199         }
200     }
201
202
203     private void addForEach(List<RecordColumn> subInputs, List<RecordColumn> subOutputs) throws
204         InvalidDataPartException {
205             numberMissingInputs = 1;
206             numberMissingOutputs = 0;
207             DataPartParameter p = subDataFlowPanel.addNewDataPartParameter("Item", typeAuto, typeAuto,
208                 (x) -> true, true);
209             this.subInputs.add(p);
210             p.setPointBoxLocation(new Point(50, 40));
211
212             functionalImpl = new FunctionalForEach();
213
214             if (subInputs != null) {
215                 RecordColumn in = subInputs.get(0);
216                 p.changeType(in.getType());
217                 p.changeName(in.getId());
218             }
219         }
220
221
222     private void addAccumulate(List<RecordColumn> subInputs, List<RecordColumn> subOutputs) throws
223         InvalidDataPartException {
224             numberMissingInputs = 2;
225             numberMissingOutputs = 1;
226             DataPartParameter p1 = subDataFlowPanel.addNewDataPartParameter("Item", typeAuto, typeAuto
227                 , (x) -> true, true);
228             p1.setPointBoxLocation(new Point(50, 40));
```

```
218     DataPartParameter p2 = subDataFlowPanel.addNewDataPartParameter("Accum", typeAuto,
219     typeAuto, (x) -> true, true);
220     p2.setPointBoxLocation(new Point(150, 40));
221     this.subInputs.add(p1);
222     this.subInputs.add(p2);
223     DataPartResult r = subDataFlowPanel.addNewDataPartResult("Accum", typeAuto, typeAuto, (x)
224     -> true, true);
225     this.subOutputs.add(r);
226     r.setPointBoxLocation(new Point(100, 200));
227     functionalImpl = new FunctionalAccumulate();
228
229     setOutputArrowListener("", r);
230     r.setInputArrowListeners(Arrays.asList(r));
231
232     if (subInputs != null) {
233         RecordColumn in = subInputs.get(0);
234         p1.changeType(in.getType());
235         p1.changeName(in.getId());
236         in = subInputs.get(1);
237         p2.changeType(in.getType());
238         p2.changeName(in.getId());
239         RecordColumn out = subOutputs.get(0);
240         r.changeName(out.getId());
241     }
242
243     private void addDataFlowPanelIO(FunctionalType type, List<RecordColumn> subInputs,
244     List<RecordColumn> subOutputs)
245     throws InvalidDataPartException {
246     switch (type) {
247     case MAP:
248         addMap(subInputs, subOutputs);
249         break;
250     case FILTER:
251         addFilter(subInputs, subOutputs);
252         break;
253     case JOIN:
254         addJoin(subInputs, subOutputs);
255         break;
256     case FOREACH:
257         addForEach(subInputs, subOutputs);
258         break;
259     case ACCUMULATE:
260         addAccumulate(subInputs, subOutputs);
261         break;
262     default:
263         throw new IllegalArgumentException("unexpected type " + type);
264     }
265
266     @Override
267     public boolean canFail() {
268         return subDataFlowPanel.canFail();
269     }
270
```

```
271     @Override
272     public void doubleClick(MainWindow mainWindow, Window parent) {
273         mainWindow.openExplorerItem(subDataFlowPanel);
274     }
275
276     @Override
277     public boolean markDeletedIfAllowed() {
278         mainWindow.removeExplorerItem(subDataFlowPanel);
279         return true;
280     }
281
282     @Override
283     public boolean addParameterIfAllowed(DataPartParameter param) {
284         if (numberMissingInputs > 0) {
285             numberMissingInputs -= 1;
286             return true;
287         }
288         JOptionPane.showMessageDialog(mainWindow, "Insert new input is not allowed here.",
289             "Unable to insert", JOptionPane.ERROR_MESSAGE);
290         return false;
291     }
292
293     @Override
294     public boolean addResultIfAllowed(DataPartResult result) {
295         if (numberMissingOutputs > 0) {
296             numberMissingOutputs -= 1;
297             return true;
298         }
299         JOptionPane.showMessageDialog(mainWindow, "Insert new output is not allowed here.",
300             "Unable to insert", JOptionPane.ERROR_MESSAGE);
301         return false;
302     }
303
304     @Override
305     public boolean removeParameterIfAllowed(DataPartParameter param) {
306         JOptionPane.showMessageDialog(mainWindow, "Delete input is not allowed here.",
307             "Unable to delete", JOptionPane.ERROR_MESSAGE);
308         return false;
309     }
310
311     @Override
312     public boolean removeResultIfAllowed(DataPartResult result) {
313         JOptionPane.showMessageDialog(mainWindow, "Delete output is not allowed here.",
314             "Unable to delete", JOptionPane.ERROR_MESSAGE);
315         return false;
316     }
317
318     @Override
319     public DataFlowPanel getSubDataFlowPanel() {
320         return subDataFlowPanel;
321     }
322
323     @Override
324     public boolean isParameterTypeConnected(String param) {
325         return functionalImpl.isParamaterTypeConnected(param);
```

```
326     }
327
328     @Override
329     public boolean isResultTypeConnected(String res) {
330         return functionalImpl.isResultTypeConnected(res);
331     }
332
333     @Override
334     public void disconnectInputTypeIfConnected(String name) {
335         functionalImpl.disconnectInputTypeIfConnected(name);
336     }
337
338     @Override
339     public void disconnectOutputTypeIfConnected(String name) {
340         functionalImpl.disconnectOutputTypeIfConnected(name);
341     }
342
343     @Override
344     public void changeOutputPortType(String oldName, Type newType) {
345         functionalImpl.changeOutputPortType(oldName, newType);
346     }
347
348     @Override
349     public void changeInputPortType(String oldName, Type newType) {
350         functionalImpl.changeInputPortType(oldName, newType);
351     }
352
353     @Override
354     public void changeInputPortName(String oldName, String newName) {
355         functionalImpl.changeInputPortName(oldName, newName);
356     }
357
358     @Override
359     public void changeOutputPortName(String oldName, String newName) {
360         functionalImpl.changeOutputPortName(oldName, newName);
361     }
362
363     private class FunctionalMap implements IFunctional {
364         public FunctionalMap() {
365             FunctionalListener ls = new FunctionalListener();
366             setInputArrowListeners(Arrays.asList(ls));
367             subInputs.get(0).setOutputArrowListener("", ls);
368         }
369
370         @Override
371         public boolean isParamaterTypeConnected(String param) {
372             return DataPartFunctional.super.isParameterTypeConnected("");
373         }
374
375         @Override
376         public boolean isResultTypeConnected(String res) {
377             return DataPartFunctional.super.isResultTypeConnected("");
378         }
379
380         @Override
```

```
381     public void disconnectInputTypeIfConnected(String name) {
382         DataPartFunctional.super.disconnectInputTypeIfConnected("");
383     }
384
385     @Override
386     public void disconnectOutputTypeIfConnected(String name) {
387         DataPartFunctional.super.disconnectOutputTypeIfConnected("");
388     }
389
390     @Override
391     public void changeOutputPortType(String oldName, Type newType) {
392         DataPartFunctional.super.changeOutputPortType("", new TypeList(newType));
393     }
394
395     @Override
396     public void changeInputPortType(String oldName, Type newType) {
397         DataPartFunctional.super.changeInputPortType("", new TypeList(newType));
398     }
399
400     @Override
401     public void changeInputPortName(String oldName, String newName) {}
402     @Override
403     public void changeOutputPortName(String oldName, String newName) {}
404
405     private class FunctionalListener implements TypeAutoInputListener, TypeAutoOutputListener
406     {
407         @Override
408         public void autoTypeInputChanged(Type newType) throws TypeAutoChangeException {
409             getDataFlowPanel().allowTypeChanges(1);
410             subInputs.get(0).changeType(((TypeList)newType).getChildType());
411         }
412
413         @Override
414         public void autoTypeInputRemoved() {
415             if (autoTypeInputMayChange()) {
416                 getDataFlowPanel().allowTypeChanges(1);
417                 subInputs.get(0).changeType(typeAuto);
418             }
419         }
420
421         @Override
422         public boolean autoTypeInputMayChange() {
423             return !subInputs.get(0).isOutputPortConnected(new Port("", new TypeUnknown(""))));
424         }
425
426         @Override
427         public void autoTypeOutputRemoved() {
428             if (!isInputPortConnected(new Port("", new TypeUnknown("")))) {
429                 autoTypeInputRemoved();
430             }
431         }
432     }
433
434     private class FunctionalFilter implements IFunctional {
```

```
435     public FunctionalFilter() {
436         FunctionalListener ls = new FunctionalListener();
437         setInputArrowListeners(Arrays.asList(ls));
438         subInputs.get(0).setOutputArrowListener("", ls);
439         setOutputArrowListener("", ls);
440     }
441
442     @Override
443     public boolean isParamaterTypeConnected(String param) {
444         return DataPartFunctional.super.isParameterTypeConnected("") ||
445                DataPartFunctional.super.isResultTypeConnected("");
446     }
447
448     @Override
449     public boolean isResultTypeConnected(String res) {
450         return false;
451     }
452
453     @Override
454     public void disconnectInputTypeIfConnected(String name) {
455         DataPartFunctional.super.disconnectInputTypeIfConnected("");
456         DataPartFunctional.super.disconnectOutputTypeIfConnected("");
457     }
458
459     @Override
460     public void disconnectOutputTypeIfConnected(String name) {
461         DataPartFunctional.super.disconnectOutputTypeIfConnected("");
462     }
463
464     @Override
465     public void changeInputPortType(String oldName, Type newType) {
466         DataPartFunctional.super.changeInputPortType("", newList(newType));
467         DataPartFunctional.super.changeOutputPortType("", newList(newType));
468     }
469
470     @Override
471     public void changeOutputPortType(String oldName, Type newType) {}
472     @Override
473     public void changeInputPortName(String oldName, String newName) {}
474     @Override
475     public void changeOutputPortName(String oldName, String newName) {}
476
477     private class FunctionalListener implements TypeAutoInputListener, TypeAutoOutputListener
{
478         @Override
479         public void autoTypeInputChanged(Type newType) throws TypeAutoChangeException {
480             getDataFlowPanel().allowTypeChanges(1);
481             subInputs.get(0).changeType(((TypeList)newType).getChildType());
482         }
483
484         @Override
485         public void autoTypeInputRemoved() {
486             if (autoTypeInputMayChange()) {
487                 getDataFlowPanel().allowTypeChanges(1);
488                 subInputs.get(0).changeType(typeAuto);
489             }
490         }
491     }
492 }
```

```
489         }
490     }
491
492     @Override
493     public boolean autoTypeInputMayChange() {
494         return !subInputs.get(0).isOutputPortConnected(new Port("", new TypeUnknown("")))
495             &&
496             !isOutputPortConnected(new Port("", new TypeUnknown(""))));
497     }
498
499     @Override
500     public void autoTypeOutputRemoved() {
501         if (!isInputPortConnected(new Port("", new TypeUnknown("")))) {
502             autoTypeInputRemoved();
503         }
504     }
505 }
506
507 private class FunctionalJoin implements IFunctional, TypeAutoOutputListener {
508     private Map<String, String> inputMap = new HashMap<>();
509     private Map<String, Type> inputTypes = new HashMap<>();
510     private FunctionalListener ls0;
511     private FunctionalListener ls1;
512
513     public FunctionalJoin() {
514         inputMap.put("Left_item", "Left");
515         inputMap.put("Right_item", "Right");
516         inputTypes.put(joinInputPorts[0].getName(), joinInputPorts[0].getType());
517         inputTypes.put(joinInputPorts[1].getName(), joinInputPorts[1].getType());
518
519         ls0 = new FunctionalListener(0);
520         ls1 = new FunctionalListener(1);
521         setInputArrowListeners(Arrays.asList(ls0, ls1));
522         subInputs.get(0).setOutputArrowListener("", ls0);
523         subInputs.get(1).setOutputArrowListener("", ls1);
524         setOutputArrowListener("", this);
525     }
526
527     @Override
528     public boolean isParamaterTypeConnected(String param) {
529         String realName = inputMap.get(param);
530         return DataPartFunctional.super.isParameterTypeConnected(realName) ||
531             DataPartFunctional.super.isResultTypeConnected("");
532     }
533
534     @Override
535     public boolean isResultTypeConnected(String res) {
536         return false;
537     }
538
539     @Override
540     public void disconnectInputTypeIfConnected(String name) {
541         String realName = inputMap.get(name);
542         DataPartFunctional.super.disconnectInputTypeIfConnected(realName);
```

```
543         DataPartFunctional.super.disconnectOutputTypeIfConnected("");
544     }
545
546     @Override
547     public void disconnectOutputTypeIfConnected(String name) {
548         DataPartFunctional.super.disconnectOutputTypeIfConnected("");
549     }
550
551     @Override
552     public void changeOutputPortType(String outputName, Type newType) {}
553
554     private Type getJoinOutputType(TypeList left, TypeList right) {
555         return new TypeList(DataPartUtil.getAugmentType(left.getChildType(), "Left", right.
556         getChildType(), "Right"));
557     }
558
559     @Override
560     public void changeInputPortType(String inputName, Type newType) {
561         newType = new TypeList(newType);
562         String realName = inputMap.get(inputName);
563         inputTypes.put(realName, newType);
564         DataPartFunctional.super.changeInputPortType(realName, newType);
565         Type leftType = inputTypes.get("Left");
566         Type rightType = inputTypes.get("Right");
567         if (!leftType.isConcrete(true) || !rightType.isConcrete(true))
568             return;
569         Type outputType = getJoinOutputType((TypeList)leftType, (TypeList)rightType);
570         DataPartFunctional.super.changeOutputPortType("", outputType);
571     }
572
573     @Override
574     public void changeInputPortName(String oldName, String newName) {
575         String s = inputMap.remove(oldName);
576         inputMap.put(newName, s);
577     }
578
579     @Override
580     public void changeOutputPortName(String oldName, String newName) {}
581
582     private class FunctionalListener implements TypeAutoInputListener, TypeAutoOutputListener
583     {
584         int index;
585
586         private String getName() {
587             switch (index) {
588                 case 0:
589                     return "Left";
590                 default:
591                     return "Right";
592             }
593         }
594
595         public FunctionalListener(int i) {
596             index = i;
597         }
598     }
599 }
```

```
596
597     @Override
598     public void autoTypeInputChanged(Type newType) throws TypeAutoChangeException {
599         getDataFlowPanel().allowTypeChanges(1);
600         subInputs.get(index).changeType(((TypeList)newType).getChildType());
601     }
602
603     @Override
604     public void autoTypeInputRemoved() {
605         if (autoTypeInputMayChange()) {
606             getDataFlowPanel().allowTypeChanges(1);
607             subInputs.get(index).changeType(typeAuto);
608         }
609     }
610
611     @Override
612     public boolean autoTypeInputMayChange() {
613         return !subInputs.get(index).isOutputPortConnected(new Port("", new TypeUnknown("")))
614             && !isOutputPortConnected(new Port("", new TypeUnknown(""))));
615     }
616
617     @Override
618     public void autoTypeOutputRemoved() {
619         if (!isInputPortConnected(new Port(getName(), new TypeUnknown("")))) {
620             autoTypeInputRemoved();
621         }
622     }
623 }
624
625     @Override
626     public void autoTypeOutputRemoved() {
627         ls0.autoTypeOutputRemoved();
628         ls1.autoTypeOutputRemoved();
629     }
630 }
631
632     private class FunctionalForEach implements IFunctional {
633         public FunctionalForEach() {
634             FunctionalListener ls = new FunctionalListener();
635             setInputArrowListeners(Arrays.asList(ls));
636             subInputs.get(0).setOutputArrowListener("", ls);
637         }
638         @Override
639         public boolean isParamaterTypeConnected(String param) {
640             return DataPartFunctional.super.isParameterTypeConnected("");
641         }
642
643         @Override
644         public boolean isResultTypeConnected(String res) {
645             return false;
646         }
647
648         @Override
649         public void disconnectInputTypeIfConnected(String name) {
```

```
650         DataPartFunctional.super.disconnectInputTypeIfConnected("");
651     }
652
653     @Override
654     public void disconnectOutputTypeIfConnected(String name) {}
655     @Override
656     public void changeOutputPortType(String oldName, Type newType) {}
657
658     @Override
659     public void changeInputPortType(String oldName, Type newType) {
660         DataPartFunctional.super.changeInputPortType("", new TypeList(newType));
661     }
662
663     @Override
664     public void changeInputPortName(String oldName, String newName) {}
665     @Override
666     public void changeOutputPortName(String oldName, String newName) {}
667
668     private class FunctionalListener implements TypeAutoInputListener, TypeAutoOutputListener
669     {
670         @Override
671         public void autoTypeInputChanged(Type newType) throws TypeAutoChangeException {
672             getDataFlowPanel().allowTypeChanges(1);
673             subInputs.get(0).changeType(((TypeList)newType).getChildType());
674         }
675
676         @Override
677         public void autoTypeInputRemoved() {
678             if (autoTypeInputMayChange()) {
679                 getDataFlowPanel().allowTypeChanges(1);
680                 subInputs.get(0).changeType(typeAuto);
681             }
682         }
683
684         @Override
685         public boolean autoTypeInputMayChange() {
686             return !subInputs.get(0).isOutputPortConnected(new Port("", new TypeUnknown(""))));
687         }
688
689         @Override
690         public void autoTypeOutputRemoved() {
691             if (!isInputPortConnected(new Port("", new TypeUnknown("")))) {
692                 autoTypeInputRemoved();
693             }
694         }
695     }
696
697     private class FunctionalAccumulate implements IFunctional {
698         private Map<String, String> inputMap = new HashMap<>();
699         private Map<String, Type> inputTypes = new HashMap<>();
700
701         public FunctionalAccumulate() {
702             inputMap.put("Item", "List");
703             inputMap.put("Accum", "Accum");
```

```
704     inputTypes.put("Item", accumInputPorts[0].getType());
705     inputTypes.put("Accum", accumInputPorts[1].getType());
706
707     FunctionalListenerList lsList = new FunctionalListenerList();
708     FunctionalListenerAcc lsAcc = new FunctionalListenerAcc();
709     setInputArrowListeners(Arrays.asList(lsList, lsAcc));
710     subInputs.get(0).setOutputArrowListener("", lsList);
711     subInputs.get(1).setOutputArrowListener("", lsAcc);
712 }
713
714 @Override
715 public boolean isParamaterTypeConnected(String param) {
716     String realName = inputMap.get(param);
717     if (realName.equals("Accum")) {
718         return DataPartFunctional.super.isParameterTypeConnected(realName) ||
719             DataPartFunctional.super.isResultTypeConnected("");
720     } else {
721         return DataPartFunctional.super.isParameterTypeConnected(realName);
722     }
723 }
724
725 @Override
726 public boolean isResultTypeConnected(String res) {
727     return DataPartFunctional.super.isResultTypeConnected("") ||
728         DataPartFunctional.super.isParameterTypeConnected("Accum");
729 }
730
731 @Override
732 public void disconnectInputTypeIfConnected(String name) {
733     String realName = inputMap.get(name);
734     DataPartFunctional.super.disconnectInputTypeIfConnected(realName);
735     if (realName.equals("Accum")) {
736         DataPartFunctional.super.disconnectOutputTypeIfConnected("");
737     }
738 }
739
740 @Override
741 public void disconnectOutputTypeIfConnected(String name) {
742     DataPartFunctional.super.disconnectInputTypeIfConnected("Accum");
743     DataPartFunctional.super.disconnectOutputTypeIfConnected("");
744 }
745
746 @Override
747 public void changeOutputPortType(String oldName, Type newType) {
748     DataPartFunctional.super.changeInputPortType("Accum", newType);
749     DataPartFunctional.super.changeOutputPortType("", newType);
750     subInputs.get(1).setType(newType);
751     subInputs.get(1).changeOutputPortType("", newType);
752 }
753
754 @Override
755 public void changeInputPortType(String oldName, Type newType) {
756     String realName = inputMap.get(oldName);
757     if (realName.equals("Accum")) {
758         DataPartFunctional.super.changeInputPortType(realName, newType);
```

```
759         DataPartFunctional.super.changeOutputPortType("", newType);
760         subOutputs.get(0).setType(newType);
761         subOutputs.get(0).changeInputPortType("", newType);
762     } else {
763         DataPartFunctional.super.changeInputPortType(realName, new TypeList(newType));
764     }
765 }
766
767 @Override
768 public void changeInputPortName(String oldName, String newName) {
769     String s = inputMap.remove(oldName);
770     inputMap.put(newName, s);
771 }
772
773 @Override
774 public void changeOutputPortName(String oldName, String newName) {}
775
776     private class FunctionalListenerList implements TypeAutoInputListener,
777     TypeAutoOutputListener {
778
779         @Override
780         public void autoTypeInputChanged(Type newType) throws TypeAutoChangeException {
781             getDataFlowPanel().allowTypeChanges(1);
782             subInputs.get(0).changeType(((TypeList)newType).getChildType());
783         }
784
785         @Override
786         public void autoTypeInputRemoved() {
787             if (autoTypeInputMayChange()) {
788                 getDataFlowPanel().allowTypeChanges(1);
789                 subInputs.get(0).changeType(typeAuto);
790             }
791         }
792
793         @Override
794         public boolean autoTypeInputMayChange() {
795             return !subInputs.get(0).isOutputPortConnected(new Port("", new TypeUnknown(""))));
796         }
797
798         @Override
799         public void autoTypeOutputRemoved() {
800             if (!isInputPortConnected(new Port("List", new TypeUnknown("")))) {
801                 autoTypeInputRemoved();
802             }
803         }
804
805         private class FunctionalListenerAcc implements TypeAutoInputListener,
806         TypeAutoOutputListener {
807
808             @Override
809             public void autoTypeInputChanged(Type newType) throws TypeAutoChangeException {
810                 getDataFlowPanel().allowTypeChanges(2);
811                 subInputs.get(1).changeType(newType);
812                 subOutputs.get(0).changeType(newType);
813             }
814
815         }
816
817     }
818 }
```

```

812     @Override
813     public void autoTypeInputRemoved() {
814         if (autoTypeInputMayChange()) {
815             getDataFlowPanel().allowTypeChanges(2);
816             subInputs.get(1).changeType(typeAuto);
817             subOutputs.get(0).changeType(typeAuto);
818         }
819     }
820
821     @Override
822     public boolean autoTypeInputMayChange() {
823         return !subInputs.get(1).isOutputPortConnected(new Port("", new TypeUnknown("")))
824             &&
825             !isOutputPortConnected(new Port("", new TypeUnknown(""))) &&
826             !subOutputs.get(0).isInputPortConnected(new Port("", new TypeUnknown("")));
827         ;
828     }
829     @Override
830     public void autoTypeOutputRemoved() {
831         if (!isInputPortConnected(new Port("Accum", new TypeUnknown("")))) {
832             autoTypeInputRemoved();
833         }
834     }
835 }
836
837 private static interface IFunctional {
838     public boolean isParamaterTypeConnected(String param);
839     public boolean isResultTypeConnected(String res);
840     public void disconnectInputTypeIfConnected(String name);
841     public void disconnectOutputTypeIfConnected(String name);
842     public void changeOutputPortType(String oldName, Type newType);
843     public void changeInputPortType(String oldName, Type newType);
844     public void changeInputPortName(String oldName, String newName);
845     void changeOutputPortName(String oldName, String newName);
846 }
847
848     public FunctionalType getFunctionalType() {
849         return functionalType;
850     }
851 }
```

LISTING E.93: gui/dataflow/DataPartFunctional.java

```

1 package gui.dataflow;
2
3 import java.awt.Graphics;
4 import java.awt.Window;
5 import java.util.Arrays;
6 import java.util.Map;
7
8 import lang.type.Type;
9 import lang.type.TypeAuto;
10 import lang.type.TypeAutoChangeException;
```

```
11 import lang.type.TypeAutoInputListener;
12 import lang.type.TypeCode;
13 import lang.type.TypeList;
14 import lang.type.TypeRecord;
15 import gui.DataFlowPanel;
16 import gui.MainWindow;
17
18 public class DataPartCast extends DataPart implements TypeAutoInputListener {
19     private boolean canCastFail = false;
20     private Type originalCastFrom;
21     private Type castFrom;
22     private Type castTo;
23
24     private static String getCastTitle(Type from, Type to) {
25         return from.getShortDescription() + " \u2192 " + to.getShortDescription();
26     }
27
28     private boolean validCastFromNumber(Type to) {
29         while (to.getTypeCode() == TypeCode.AUTO)
30             to = ((TypeAuto)to).getType();
31         switch (to.getTypeCode()) {
32             case TIME:
33             case NUMBER:
34             case STRING:
35                 return true;
36             default:
37                 return false;
38         }
39     }
40
41     private boolean validCastFromString(Type to) {
42         while (to.getTypeCode() == TypeCode.AUTO)
43             to = ((TypeAuto)to).getType();
44         switch (to.getTypeCode()) {
45             case STRING:
46                 return true;
47             case BOOL:
48             case NUMBER:
49                 canCastFail = true;
50                 return true;
51             default:
52                 return false;
53         }
54     }
55
56     private boolean validCastFromBool(Type to) {
57         while (to.getTypeCode() == TypeCode.AUTO)
58             to = ((TypeAuto)to).getType();
59         switch (to.getTypeCode()) {
60             case BOOL:
61             case STRING:
62                 return true;
63             default:
64                 return false;
65         }
66     }
67 }
```

```
66     }
67
68     private boolean validCastFromTime(Type to) {
69         while (to.getTypeCode() == TypeCode.AUTO)
70             to = ((TypeAuto)to).getType();
71         switch (to.getTypeCode()) {
72             case TIME:
73             case NUMBER:
74                 return true;
75             default:
76                 return false;
77         }
78     }
79
80     private boolean validCastFromList(TypeList from, Type to) {
81         while (to.getTypeCode() == TypeCode.AUTO)
82             to = ((TypeAuto)to).getType();
83         if (to.getTypeCode() != TypeCode.LIST)
84             return false;
85         return validCast(from.getChildType(), ((TypeList)to).getChildType());
86     }
87
88     private boolean validCastFromRecord(TypeRecord from, Type to) {
89         while (to.getTypeCode() == TypeCode.AUTO)
90             to = ((TypeAuto)to).getType();
91         if (to.getTypeCode() != TypeCode.RECORD)
92             return false;
93         TypeRecord toRec = (TypeRecord)to;
94         Map<String,Type> fromMap = from.getColumns();
95         Map<String,Type> toMap = toRec.getColumns();
96         if (fromMap.size() != toMap.size())
97             return false;
98         for (String k : fromMap.keySet()) {
99             Type toType = toMap.get(k);
100            if (toType == null)
101                return false;
102            Type fromType = fromMap.get(k);
103            if (!validCast(fromType, toType))
104                return false;
105        }
106        return true;
107    }
108
109    private boolean validCast(Type from, Type to) {
110        while (from.getTypeCode() == TypeCode.AUTO)
111            from = ((TypeAuto)from).getType();
112        switch (from.getTypeCode()) {
113            case BOOL:
114                return validCastFromBool(to);
115            case LIST:
116                return validCastFromList((TypeList)from, to);
117            case NUMBER:
118                return validCastFromNumber(to);
119            case RECORD:
120                return validCastFromRecord((TypeRecord)from, to);

```

```
121     case STRING:
122         return validCastFromString(to);
123     case TIME:
124         return validCastFromTime(to);
125     case UNKNOWN:
126         return true;
127     default:
128         throw new IllegalArgumentException("invalid type cast type");
129     }
130 }
131
132 public DataPartCast(Type originalFromType, Type fromType, Type toType, DataFlowPanel dfp,
133     Graphics g)
134     throws InvalidDataPartException {
135     super(new Port[]{new Port("", fromType)}, new Port[]{new Port("", toType)},
136         getCastTitle(fromType, toType), "Cast", dfp, g);
137     if (!validCast(fromType, toType))
138         throw new InvalidDataPartException("Invalid type cast");
139     if (!validCast(originalFromType, toType))
140         throw new InvalidDataPartException("Invalid type cast");
141
142     setInputArrowListeners(Arrays.asList(this));
143     this.originalCastFrom = originalFromType;
144     castFrom = fromType;
145     castTo = toType;
146 }
147
148 public Type getFromType() {
149     return castFrom;
150 }
151
152 public Type getOriginalFromType() {
153     return originalCastFrom;
154 }
155
156 public Type getToType() {
157     return castTo;
158 }
159
160 @Override
161 public boolean canFail() {
162     return canCastFail;
163 }
164
165 @Override
166 public void doubleClick(MainWindow mainWindow, Window parent) {}
167
168 @Override
169 public boolean markDeletedIfAllowed() {
170     return true;
171 }
172
173 @Override
174 public void autoTypeInputChanged(Type newType) throws TypeAutoChangeException {
175     if (!validCast(newType, castTo))
```

```

175         throw new TypeAutoChangeException("Invalid type cast");
176     changeInputPortType("", newType);
177     castFrom = newType;
178 }
179
180 @Override
181 public void autoTypeInputRemoved() {
182     changeInputPortType("", originalCastFrom);
183     castFrom = originalCastFrom;
184 }
185
186 @Override
187 public boolean autoTypeInputMayChange() {
188     return true;
189 }
190
191 }
```

LISTING E.94: gui/dataflow/DataPartCast.java

```

1 package gui.dataflow;
2
3 import java.awt.Graphics;
4 import java.awt.Window;
5 import java.util.Arrays;
6
7 import lang.type.RecordColumn;
8 import lang.type.Type;
9 import lang.type.TypeAuto;
10 import lang.type.TypeAutoChangeException;
11 import lang.type.TypeAutoInputListener;
12 import lang.type.TypeBool;
13 import lang.type.TypeCode;
14 import lang.type.TypeRecord;
15 import lang.type.TypeUnknown;
16 import gui.DataFlowPanel;
17 import gui.MainWindow;
18
19 public class DataPartComparison extends DataPart {
20     public enum ComparisonType {LESS, GREATER, LESSEQUAL, GREATEREQUAL, EQUAL, NOTEQUAL};
21
22     private final static Port[] outputs = {new Port("", new TypeBool())};
23
24     private Type originalComparisonDataType;
25     private Type comparisonDataType;
26     private ComparisonType comparisonType;
27
28     private static Port[] getBinaryInputs(Type t) {
29         return new Port[]{new Port("!0", t), new Port("!1", t)};
30     }
31
32     private static String getComparisonTitle(ComparisonType type) {
33         switch (type) {
34             case EQUAL:
35                 return "=";
```

```
36     case GREATER:
37         return ">";
38     case GREATEREQUAL:
39         return "\u2265";
40     case LESS:
41         return "<";
42     case LESSEQUAL:
43         return "\u2264";
44     case NOTEQUAL:
45         return "\u2260";
46     default:
47         throw new RuntimeException("unexpected comparison type " + type);
48     }
49 }
50
51     private static void validateDataType(ComparisonType ctype, Type dtype)
52             throws InvalidDataPartException {
53     while (dtype.getTypeCode() == TypeCode.AUTO)
54         dtype = ((TypeAuto)dtype).getType();
55     switch (ctype) {
56     case EQUAL:
57     case NOTEQUAL:
58         switch (dtype.getTypeCode()) {
59             case LIST:
60                 break;
61             case RECORD:
62                 TypeRecord r = (TypeRecord)dtype;
63                 for (RecordColumn c : r.getColumnsInOrder())
64                     validateDataType(ctype, c.getType());
65                 return;
66             default:
67                 return;
68         }
69         break;
70     case LESS:
71     case LESSEQUAL:
72     case GREATER:
73     case GREATEREQUAL:
74         if (dtype.getTypeCode() == TypeCode.NUMBER)
75             return;
76         break;
77     default:
78         throw new RuntimeException();
79     }
80     throw new InvalidDataPartException("Cannot " + getComparisonTitle(ctype) +
81             " compare type '" + dtype.getShortDescription() + "'");
82 }
83
84     private class AutoListener implements TypeAutoInputListener {
85         private int index;
86
87         private AutoListener(int index) {
88             this.index = index;
89         }
90     }
```

```
91     private String getName() {
92         return "!" + index;
93     }
94
95     private String getOtherName() {
96         return "!" + ((index+1)%2);
97     }
98
99     @Override
100    public void autoTypeInputChanged(Type newType) throws TypeAutoChangeException {
101        try {
102            validateDataType(comparisonType, newType);
103        } catch (InvalidDataPartException ex) {
104            throw new TypeAutoChangeException(ex.getMessage());
105        }
106        if (comparisonDataType.equals(newType))
107            return;
108        disconnectInputTypeIfConnected(getOtherName());
109        comparisonDataType = newType;
110        changeInputPortType(getName(), newType);
111        changeInputPortType(getOtherName(), newType);
112    }
113
114    @Override
115    public void autoTypeInputRemoved() {
116        if (isInputPortConnected(new Port(getOtherName(), new TypeUnknown("")))) {
117            return;
118        }
119        changeInputPortType(getName(), originalComparisonDataType);
120        changeInputPortType(getOtherName(), originalComparisonDataType);
121        comparisonDataType = originalComparisonDataType;
122    }
123
124    @Override
125    public boolean autoTypeInputMayChange() {
126        return !isInputPortConnected(new Port(getOtherName(), new TypeUnknown("")));
127    }
128}
129
130    public DataPartComparison(ComparisonType ctype, Type origDType, Type dtype, DataFlowPanel dfp,
131        Graphics g)
132        throws InvalidDataPartException {
133        super(getBinaryInputs(dtype), outputs, getComparisonTitle(ctype), "Comparison", dfp, g);
134        setInputArrowListeners(Arrays.asList(new AutoListener(0), new AutoListener(1)));
135        validateDataType(ctype, dtype);
136        validateDataType(ctype, origDType);
137        originalComparisonDataType = origDType;
138        comparisonDataType = dtype;
139        comparisonType = ctype;
140    }
141
142    public Type dataType() {
143        return comparisonDataType;
144    }
```

```
145     public Type getOriginalDataType() {
146         return originalComparisonDataType;
147     }
148
149     public ComparisonType getComparisonType() {
150         return comparisonType;
151     }
152
153     @Override
154     public boolean canFail() {
155         return false;
156     }
157
158     @Override
159     public void doubleClick(MainWindow mainWindow, Window parent) {
160     }
161
162     @Override
163     public boolean markDeletedIfAllowed() {
164         return true;
165     }
166
167 }
```

LISTING E.95: gui/dataflow/DataPartComparison.java

```
1 package gui;
2
3 import java.awt.BorderLayout;
4 import java.awt.Color;
5 import java.awt.Component;
6 import java.awt.Dimension;
7 import java.awt.FlowLayout;
8 import java.awt.Graphics;
9 import java.awt.Graphics2D;
10 import java.awt.GridBagConstraints;
11 import java.awt.GridBagLayout;
12 import java.awt.GridLayout;
13 import java.awt.event.ActionEvent;
14 import java.awt.event.ActionListener;
15 import java.awt.event.KeyAdapter;
16 import java.awt.event.KeyEvent;
17 import java.awt.event.KeyListener;
18 import java.awt.event.MouseAdapter;
19 import java.awt.event.MouseEvent;
20 import java.awt.font.TextLayout;
21 import java.awt.geom.Rectangle2D;
22 import java.util.ArrayList;
23 import java.util.List;
24
25 import javax.swing.BorderFactory;
26 import javax.swingBoxLayout;
27 import javax.swing.JMenuItem;
28 import javax.swing.JPanel;
29 import javax.swing.JPopupMenu;
```

```
30 import javax.swing.JScrollPane;
31 import javax.swing.JTextArea;
32 import javax.swing.text.DefaultCaret;
33
34 import lang.constant.Constant;
35 import lang.constant.ConstantBool;
36 import lang.constant.ConstantList;
37 import lang.constant.ConstantNumber;
38 import lang.constant.ConstantRecord;
39 import lang.constant.ConstantString;
40 import lang.constant.ConstantTime;
41 import lang.constant.InvalidConstantException;
42 import lang.constant.RecordValue;
43 import lang.type.RecordColumn;
44 import lang.type.Type;
45 import lang.type.TypeChangedListener;
46 import lang.type.TypeCode;
47 import lang.type.TypeError;
48 import lang.type.TypeList;
49 import lang.type.TypeRecord;
50 import lang.type.TypeUnknown;
51 import util.StringUtil;
52 import util.UtilFont;
53 import util.UtilPrompt;
54 import util.UtilScrollPane;
55
56 @SuppressWarnings("serial")
57 public class ValuePanel extends JPanel implements TypeChangedListener {
58     private JPanel currentBorderPanel;
59     private ValueJPanel currentPanel;
60     private Component parent;
61     private Graphics initialGraphics;
62
63     ValuePanel(ConstantDefiner parent, Graphics g) {
64         this(parent.getTypeSelector().getType(), null, parent, g);
65     }
66
67     public ValuePanel(Type type, Constant value, Component parent, Graphics g) {
68         super(new FlowLayout(FlowLayout.LEADING, 10, 10));
69
70         initialGraphics = g;
71
72         this.parent = parent;
73         currentPanel = getPanel(type, value);
74         currentBorderPanel = new JPanel();
75         currentBorderPanel.add(currentPanel);
76         currentBorderPanel.setBorder(BorderFactory.createTitledBorder("Enter Value"));
77
78         add(currentBorderPanel);
79     }
80
81     public Constant getValue() throws InvalidConstantException {
82         return currentPanel.getValue();
83     }
84
```

```
85     @Override
86     public void typeChanged(Type newType) {
87         remove(currentBorderPanel);
88
89         currentPanel = getPanel(newType, null);
90         currentBorderPanel = new JPanel();
91         currentBorderPanel.add(currentPanel);
92         currentBorderPanel.setBorder(BorderFactory.createTitledBorder("Enter Value"));
93
94         add(currentBorderPanel);
95         parent.revalidate();
96         parent.repaint();
97     }
98
99     public ValueJPanel getPanel(Type type, Constant value) {
100        switch (type.getTypeCode()) {
101            case NUMBER:
102                return getNumberPanel((ConstantNumber)value);
103            case BOOL:
104                return getBoolPanel((ConstantBool)value);
105            case TIME:
106                return getTimePanel((ConstantTime)value);
107            case STRING:
108                return getStringPanel((ConstantString)value);
109            case RECORD:
110                return getRowPanel((TypeRecord)type, (ConstantRecord)value);
111            case LIST:
112                return getTablePanel((TypeList)type, (ConstantList)value);
113            case ERROR:
114                return getErrorPanel((TypeError)type);
115            case UNKNOWN:
116                return getUnknownPanel((TypeUnknown)type);
117            default:
118                break;
119        }
120        return null;
121    }
122
123    abstract static class ValueJPanel extends JPanel {
124        private ValueJPanel() {
125            super(new GridLayout());
126        }
127        public abstract Constant getValue() throws InvalidConstantException;
128    }
129
130    final static int maxJTextAreaWidth = 180;
131    final static int maxJTextAreaHeight = 90;
132
133    private static int jTextAreaLineHeight = 0;
134    static {
135        JTextArea textArea = getResizableTextArea();
136        jTextAreaLineHeight = textArea.getPreferredSize().height;
137    }
138
139    private static JTextArea getResizableTextArea() {
```

```
140     JTextArea textArea = new JTextArea(1,8) {
141         private JTextAreaAdapter adapter;
142
143         @Override
144         public Dimension getPreferredSize() {
145             Dimension ret = super.getPreferredSize();
146             ret.width += 1;
147             return ret;
148         }
149         @Override
150         public Dimension getPreferredScrollableViewportSize() {
151             Dimension size = super.getPreferredSize();
152             if (size.height > maxJTextAreaHeight)
153                 size.height = maxJTextAreaHeight;
154             if (size.width > maxJTextAreaWidth)
155                 size.width = maxJTextAreaWidth;
156             return size;
157         }
158         @Override
159         public void addKeyListener(KeyListener k) {
160             super.addKeyListener(k);
161             if (k instanceof JTextAreaAdapter) {
162                 adapter = (JTextAreaAdapter)k;
163             }
164         }
165         @Override
166         public void setText(String t) {
167             super.setText(t);
168             adapter.handleEvent();
169             setCaretPosition(0);
170         }
171     };
172     return textArea;
173 }
174
175     private abstract static class JTextAreaAdapter extends KeyAdapter {
176         @Override
177         abstract public void keyReleased(KeyEvent e);
178         abstract public void handleEvent();
179     }
180
181     private void addJTextAreaAdapter(JTextArea text, JTextAreaAdapter a) {
182         text.addKeyListener(a);
183     }
184
185     private void addResizableTextArea(JTextArea textArea, JPanel jpanel, String hint) {
186         if (hint != null) {
187             UtilPrompt.setPrompt(hint, textArea);
188         }
189
190         DefaultCaret caret = (DefaultCaret)textArea.getCaret();
191         caret.setUpdatePolicy(DefaultCaret.ALWAYS_UPDATE);
192         textArea.setBorder(BorderFactory.createLoweredBevelBorder());
193         addJTextAreaAdapter(textArea, new JTextAreaAdapter() {
194             UtilScrollPane scroll;
```

```
195     boolean vertical = false;
196     boolean horizontal = false;
197
198     private Dimension getTextSize() {
199         String text = textArea.getText();
200
201         Graphics2D g2 = (Graphics2D)ValuePanel.this.initialGraphics;
202
203         int longestWidth = 0;
204         int lineCount = StringUtil.countSubstring(text, "\n") + 1;
205         String[] lines = text.split("\n");
206         for (int i = 0; i < lines.length; i++) {
207             if (lines[i].equals(""))
208                 continue;
209             TextLayout line = new TextLayout(lines[i], textArea.getFont(), g2.
210             getFontRenderContext());
211             Rectangle2D r = line.getBounds();
212             longestWidth = Math.max(longestWidth, (int)Math.ceil(r.getWidth()));
213         }
214         int height = jTextAreaLineHeight * lineCount;
215         return new Dimension(longestWidth, height);
216     }
217
218     @Override
219     public void handleEvent() {
220         Dimension size = getTextSize();
221
222         if (scroll != null) {
223             size.width += 20;
224             size.height += 20;
225         }
226         if (size.width > maxJTextAreaWidth && size.height > maxJTextAreaHeight) {
227             if (scroll == null || !vertical || !horizontal) {
228                 if (scroll != null) {
229                     scroll.remove(textArea);
230                     jPanel.remove(scroll);
231                 }
232                 jPanel.remove(textArea);
233                 scroll = new UtilScrollPane(textArea,
234                     JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
235                     JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
236                 scroll.setBorder(BorderFactory.createLoweredBevelBorder());
237                 jPanel.add(scroll);
238                 textArea.requestFocus();
239                 vertical = true;
240                 horizontal = true;
241             }
242         } else if (size.width > maxJTextAreaWidth) {
243             if (scroll == null || !vertical || horizontal) {
244                 if (scroll != null) {
245                     scroll.remove(textArea);
246                     jPanel.remove(scroll);
247                 }
248                 jPanel.remove(textArea);
249                 scroll = new UtilScrollPane(textArea,
```

```
249                     JScrollPane.VERTICAL_SCROLLBAR_NEVER,
250                     JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
251             scroll.setBorder(BorderFactory.createLoweredBevelBorder());
252             jPanel.add(scroll);
253             textArea.requestFocus();
254             vertical = true;
255             horizontal = false;
256         }
257     } else if (size.height > maxJTextAreaHeight) {
258         if (scroll == null || !horizontal || vertical) {
259             if (scroll != null) {
260                 scroll.remove(textArea);
261                 jPanel.remove(scroll);
262             }
263             jPanel.remove(textArea);
264             scroll = new UtilScrollPane(textArea,
265                     JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
266                     JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
267             scroll.setBorder(BorderFactory.createLoweredBevelBorder());
268             jPanel.add(scroll);
269             textArea.requestFocus();
270             vertical = false;
271             horizontal = true;
272         }
273     } else {
274         if (scroll != null) {
275             scroll.remove(textArea);
276             jPanel.remove(scroll);
277             jPanel.add(textArea);
278             textArea.requestFocus();
279             scroll = null;
280             vertical = false;
281             horizontal = false;
282         }
283     }
284     ValuePanel.this.revalidate();
285     ValuePanel.this.repaint();
286 }
287
288     @Override
289     public void keyReleased(KeyEvent e) {
290         handleEvent();
291     }
292 });
293 jPanel.add(textArea);
294 }
295
296     private JTextArea addResizableTextArea(JPanel jPanel, String hint) {
297         JTextArea textArea = getResizableTextArea();
298         addResizableTextArea(textArea, jPanel, hint);
299         return textArea;
300     }
301
302
303     private static class String JPanel extends Value JPanel {
```

```
304     private JTextArea textArea;
305     public void setTextArea(JTextArea a) {
306         textArea = a;
307     }
308     public Constant getValue() throws InvalidConstantException {
309         return ConstantString.parse(textArea.getText());
310     }
311 }
312 private JPanel getStringPanel(ConstantString val) {
313     StringJPanel ret = new StringJPanel();
314     JTextArea area = addResizableTextArea(ret, "String");
315     if (val != null)
316         area.setText(val.toString());
317     ret.setTextArea(area);
318     return ret;
319 }
320
321 private static class NumberJPanel extends ValueJPanel {
322     private JTextArea textArea;
323     public void setTextArea(JTextArea a) {
324         textArea = a;
325     }
326     public Constant getValue() throws InvalidConstantException {
327         return ConstantNumber.parse(textArea.getText());
328     }
329 }
330 private JPanel getNumberPanel(ConstantNumber val) {
331     NumberJPanel ret = new NumberJPanel();
332     JTextArea area = addResizableTextArea(ret, "Number");
333     if (val != null)
334         area.setText(val.toString());
335     ret.setTextArea(area);
336     return ret;
337 }
338
339 private JPanel getUnknownPanel(TypeUnknown t) {
340     ValueJPanel ret = new ValueJPanel() {
341         @Override
342         public Constant getValue() throws InvalidConstantException {
343             throw new InvalidConstantException(t.getMessage());
344         }
345     };
346     JTextArea text = addResizableTextArea(ret, "Type unknown");
347     text.setText(t.getMessage());
348     text.setEditable(false);
349     text.setFont(UtilFont.errorTextFont);
350     text.setForeground(Color.red);
351     return ret;
352 }
353
354 private JPanel getErrorPanel(TypeError t) {
355     ValueJPanel ret = new ValueJPanel() {
356         @Override
357         public Constant getValue() throws InvalidConstantException {
358             throw new InvalidConstantException("constant of 'Error' type is not allowed");
```

```
359         }
360     };
361     JTextArea text = addResizableTextArea(ret, "Error");
362     text.setText("Error");
363     text.setEditable(false);
364     text.setFont(UtilFont.errorTextFont);
365     text.setForeground(Color.red);
366     return ret;
367 }
368
369     private static class BoolJPanel extends ValueJPanel {
370         private JTextArea textArea;
371         public void setTextArea(JTextArea a) {
372             textArea = a;
373         }
374         public Constant getValue() throws InvalidConstantException {
375             return ConstantBool.parse(textArea.getText());
376         }
377     }
378     private ValueJPanel getBoolPanel(ConstantBool val) {
379         BoolJPanel ret = new BoolJPanel();
380         JTextArea area = addResizableTextArea(ret, "Boolean");
381         if (val != null)
382             area.setText(val.toString());
383         ret.setTextArea(area);
384         return ret;
385     }
386
387     private static class TimeJPanel extends ValueJPanel {
388         private JTextArea textArea;
389         public void setTextArea(JTextArea a) {
390             textArea = a;
391         }
392         public Constant getValue() throws InvalidConstantException {
393             return ConstantTime.parse(textArea.getText());
394         }
395     }
396     private ValueJPanel getTimePanel(ConstantTime val) {
397         TimeJPanel ret = new TimeJPanel();
398         JTextArea area = addResizableTextArea(ret, "Time");
399         if (val != null)
400             area.setText(val.toString());
401         ret.setTextArea(area);
402         return ret;
403     }
404
405     class RowJPanelColumn {
406         JTextArea text;
407         ValueJPanel value;
408
409         RowJPanelColumn(JTextArea a, ValueJPanel p) {
410             text = a;
411             value = p;
412         }
413     }
```

```
414
415     class RowJPanel extends ValueJPanel {
416         ArrayList<RowJPanelColumn> columns = new ArrayList<>();
417
418         @Override
419         public Constant getValue() throws InvalidConstantException {
420             ArrayList<RecordValue> vals = new ArrayList<>();
421             for (RowJPanelColumn col : columns)
422                 vals.add(new RecordValue(col.text.getText(), col.value.getValue()));
423             try {
424                 return new ConstantRecord(vals);
425             } catch (IllegalArgumentException e) {
426                 throw new InvalidConstantException(e.getMessage());
427             }
428         }
429     }
430
431     private RowJPanel getRowPanel(TypeRecord type, ConstantRecord val) {
432         RowJPanel ret = new RowJPanel();
433
434         GridBagLayout columnLayout = new GridBagLayout();
435         GridBagConstraints constraints = new GridBagConstraints();
436         ret.setLayout(columnLayout);
437
438         List<RecordColumn> cols = type.getColumnsInOrder();
439         List<RecordValue> values;
440         if (val != null)
441             values = val.getColumnsInOrder();
442         else
443             values = null;
444         int xIdx = 0;
445         for (int index = 0; index < cols.size(); index++) {
446             RecordColumn col = cols.get(index);
447             JTextArea text = addResizableTextArea(ret, "Column");
448             text.setText(col.getId());
449             text.setEditable(false);
450             text.setBackground(new Color(0xe0, 0xff, 0xff));
451             ValueJPanel child = getPanel(col.getType(), values == null ? null : values.get(index).
452                 getConstant());
453
454             constraints.fill = GridBagConstraints.BOTH;
455             constraints.weightx = 0;
456             constraints.weighty = 0;
457             constraints.gridx = xIdx;
458             constraints.gridy = 0;
459             constraints.gridwidth = 1;
460             constraints.gridheight = 1;
461             JPanel dummy = new JPanel(new BorderLayout());
462             columnLayout.setConstraints(dummy, constraints);
463             dummy.add(text, BorderLayout.CENTER);
464             ret.add(dummy);
465
466             constraints.fill = GridBagConstraints.BOTH;
467             constraints.weightx = 100;
468             constraints.weighty = 100;
```

```
468     constraints.gridx = xIdx;
469     constraints.gridy = 1;
470     constraints.gridwidth = 1;
471     constraints.gridheight = 1;
472     dummy = new JPanel(new BorderLayout());
473     columnLayout.setConstraints(dummy, constraints);
474     dummy.add(child, BorderLayout.CENTER);
475     ret.add(dummy);
476
477     ret.columns.add(new RowJPanelColumn(text, child));
478     xIdx++;
479 }
480
481 return ret;
482 }
483
484 class TableEntryJPanel extends JPanel {
485     int rowNum;
486     TypeList typeTable;
487     JTextArea numberTextArea;
488     Component child;
489     TableEntryJPanel(int rowNum, TypeList type, boolean enabled, Constant val) {
490         this.rowNum = rowNum;
491         this.typeTable = type;
492
493         GridBagLayout layout = new GridBagLayout();
494         GridBagConstraints constraints = new GridBagConstraints();
495         setLayout(layout);
496
497         constraints.fill = GridBagConstraints.VERTICAL;
498         constraints.weightx = 0;
499         constraints.weighty = 100;
500         constraints.gridx = 0;
501         constraints.gridy = 0;
502         constraints.gridwidth = 1;
503         constraints.gridheight = 1;
504
505         numberTextArea = getResizableTextArea();
506         layout.setConstraints(numberTextArea, constraints);
507         addResizableTextArea(numberTextArea, this, "Row number");
508         numberTextArea.setColumns(2);
509         numberTextArea.setText(Integer.toString(rowNum));
510         numberTextArea.setEditable(false);
511         numberTextArea.setBackground(new Color(0xCC, 0xFF, 0xCC));
512
513         constraints.fill = GridBagConstraints.BOTH;
514         constraints.weightx = 100;
515         constraints.weighty = 100;
516         constraints.gridx = 1;
517         constraints.gridy = 0;
518         constraints.gridwidth = 1;
519         constraints.gridheight = 1;
520         if (enabled) {
521             ValueJPanel p = getPanel(typeTable.getChildType(), val);
522             layout.setConstraints(p, constraints);
```

```
523         add(p);
524         child = p;
525     } else {
526         JTextArea nil = getResizableTextArea();
527         addResizableTextArea(nil, this, null);
528         layout.setConstraints(nil, constraints);
529         nil.setEnabled(false);
530         nil.setBackground(new Color(0xd3, 0xd3, 0xd3));
531         child = nil;
532     }
533 }
534 }
535
536 class TableJPanel extends ValueJPanel {
537     TableEntryJPanel firstRow;
538     ArrayList<TableEntryJPanel> rows = new ArrayList<>();
539
540     TableJPanel(TableEntryJPanel firstRow) {
541         this.firstRow = firstRow;
542         add(firstRow);
543         setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
544     }
545
546     void addFirstRow(Constant val) {
547         remove(firstRow);
548         addEnabledRow(this, 1, firstRow.typeTable, val);
549         ValuePanel.this.revalidate();
550         ValuePanel.this.repaint();
551     }
552
553     void addRow(TableEntryJPanel row, boolean addAfter, Constant val) {
554         ArrayList<TableEntryJPanel> after = new ArrayList<>();
555         int rowIndex = rows.size()-1;
556         for (; rowIndex >= 0; rowIndex--) {
557             TableEntryJPanel old = rows.get(rowIndex);
558             if (old == row) {
559                 break;
560             }
561             after.add(0, old);
562         }
563
564         if (rowIndex < 0)
565             throw new RuntimeException("unable to find expected row");
566
567         if (addAfter) {
568             rowIndex += 1;
569         } else {
570             after.add(0, rows.get(rowIndex));
571         }
572         while (rowIndex < rows.size()) {
573             remove(rows.remove(rowIndex));
574         }
575
576         addEnabledRow(this, rowIndex+1, firstRow.typeTable, val);
577     }
```

```
578     for (TableEntryJPanel p : after) {
579         rowIndex += 1;
580         p.rowNumber = rowIndex+1;
581         p.numberTextArea.setText(Integer.toString(rowIndex+1));
582         add(p);
583         rows.add(p);
584     }
585
586     ValuePanel.this.revalidate();
587     ValuePanel.this.repaint();
588 }
589
590     void addRowAfter(TableEntryJPanel row, Constant val) {
591         addRow(row, true, val);
592     }
593
594     void addRowBefore(TableEntryJPanel row, Constant val) {
595         addRow(row, false, val);
596     }
597
598     void removeRow(TableEntryJPanel row) {
599         for (int i = rows.size()-1; i >= 0; i--) {
600             TableEntryJPanel p = rows.get(i);
601             if (p == row) {
602                 rows.remove(i);
603                 remove(row);
604                 break;
605             }
606             p.rowNumber--;
607             p.numberTextArea.setText(Integer.toString(p.rowNumber));
608         }
609
610         if (rows.size() == 0)
611             add(firstRow);
612
613         ValuePanel.this.revalidate();
614         ValuePanel.this.repaint();
615     }
616
617     @Override
618     public Constant getValue() throws InvalidConstantException {
619         ArrayList<Constant> values = new ArrayList<>();
620         for (TableEntryJPanel r : rows) {
621             values.add(((ValueJPanel)r.child).getValue());
622         }
623         Type child = firstRow.typeTable.getChildType();
624         if (child.getTypeCode() != TypeCode.UNKNOWN)
625             return new ConstantList(values, firstRow.typeTable.getChildType());
626         throw new InvalidConstantException(((TypeUnknown)child).getMessage());
627     }
628 }
629
630     private void setFirstRowPopup(TableJPanel table) {
631         ActionListener a1 = new ActionListener() {
632             @Override
```

```
633     public void actionPerformed(ActionEvent e) {
634         table.addFirstRow(null);
635     }
636 };
637 JMenuItem menuItem1 = new JMenuItem("Insert row");
638 menuItem1.addActionListener(a1);
639
640 JPopupMenu popup = new JPopupMenu();
641 popup.add(menuItem1);
642
643 table.firstRow.numberTextArea.addMouseListener(new MouseAdapter() {
644     public void mousePressed(MouseEvent e) {
645         maybeShowPopup(e);
646     }
647     public void mouseReleased(MouseEvent e) {
648         maybeShowPopup(e);
649     }
650     private void maybeShowPopup(MouseEvent e) {
651         if (e.isPopupTrigger()) {
652             popup.show(e.getComponent(), e.getX(), e.getY());
653         }
654     }
655 });
656 }
657
658 private TableEntryJPanel addEnabledRow(TableJPanel table, int idx, TypeList type, Constant val
659 ) {
660     assert idx >= 0;
661
662     TableEntryJPanel row = new TableEntryJPanel(idx, type, true, val);
663
664     ActionListener a1 = new ActionListener() {
665         @Override
666         public void actionPerformed(ActionEvent e) {
667             table.addRowAfter(row, null);
668         }
669     };
670     JMenuItem menuItem1 = new JMenuItem("Insert row after");
671     menuItem1.addActionListener(a1);
672
673     ActionListener a2 = new ActionListener() {
674         @Override
675         public void actionPerformed(ActionEvent e) {
676             table.addRowBefore(row, null);
677         }
678     };
679     JMenuItem menuItem2 = new JMenuItem("Insert row before");
680     menuItem2.addActionListener(a2);
681
682     ActionListener a3 = new ActionListener() {
683         @Override
684         public void actionPerformed(ActionEvent e) {
685             table.removeRow(row);
686         }
687     };
688 }
```

```

687     JMenuItem menuItem3 = new JMenuItem("Remove row");
688     menuItem3.addActionListener(a3);
689
690     JPopupMenu popup = new JPopupMenu();
691     popup.add(menuItem1);
692     popup.add(menuItem2);
693     popup.add(menuItem3);
694
695     row.numberTextArea.addMouseListener(new MouseAdapter() {
696         public void mousePressed(MouseEvent e) {
697             maybeShowPopup(e);
698         }
699         public void mouseReleased(MouseEvent e) {
700             maybeShowPopup(e);
701         }
702         private void maybeShowPopup(MouseEvent e) {
703             if (e.isPopupTrigger()) {
704                 popup.show(e.getComponent(), e.getX(), e.getY());
705             }
706         }
707     });
708
709     table.add(row);
710     table.rows.add(row);
711
712     return row;
713 }
714
715     private JPanel getTablePanel(TypeList type, ConstantList constantList) {
716         TableEntryJPanel row = new TableEntryJPanel(0, type, false, null);
717         TableJPanel ret = new TableJPanel(row);
718         setFirstRowPopup(ret);
719         if (constantList == null)
720             return ret;
721         List<Constant> values = constantList.getValues();
722         if (values.size() == 0)
723             return ret;
724         ret.addFirstRow(values.get(0));
725         for (int i = 1; i < values.size(); i++) {
726             ret.addRowAfter(ret.rows.get(ret.rows.size()-1), values.get(i));
727         }
728         return ret;
729     }
730 }
```

LISTING E.96: gui/ValuePanel.java

```

1 package gui;
2
3 import gui.dataflow.DataPart;
4 import gui.dataflow.DataPartArithmetic;
5 import gui.dataflow.DataPartCast;
6 import gui.dataflow.DataPartComparison;
7 import gui.dataflow.DataPartConstant;
8 import gui.dataflow.DataPartFunctional;
```

```
9 import gui.dataflow.DataPartListOperation;
10 import gui.dataflow.DataPartParameter;
11 import gui.dataflow.DataPartRecordConstructor;
12 import gui.dataflow.DataPartResult;
13 import gui.dataflow.DataPartSink;
14 import gui.dataflow.DataPartStringFormat;
15 import gui.dataflow.DataPartStringOperation;
16 import gui.dataflow.DataPartSubFlow;
17 import gui.dataflow.GraphUtil;
18 import gui.dataflow.IDataPartSubFlow;
19 import gui.dataflow.InvalidDataPartException;
20 import gui.dataflow.LineConnectPart;
21 import lang.constant.Constant;
22 import lang.type.RecordColumn;
23 import lang.type.Type;
24 import lang.type.TypeConnection;
25 import lang.type.TypeRecord;
26
27 import java.awt.BasicStroke;
28 import java.awt.Color;
29 import java.awt.Dimension;
30 import java.awt.Graphics;
31 import java.awt.Graphics2D;
32 import java.awt.Point;
33 import java.awt.Rectangle;
34 import java.awt.RenderingHints;
35 import java.awt.Stroke;
36 import java.awt.event.ActionEvent;
37 import java.awt.event.ActionListener;
38 import java.awt.event.MouseAdapter;
39 import java.awt.event.MouseEvent;
40 import java.awt.event.MouseMotionListener;
41 import java.util.ArrayList;
42 import java.util.Arrays;
43 import java.util.Collections;
44 import java.util.List;
45 import java.util.function.Predicate;
46
47 import javax.swing.Icon;
48 import javax.swing.JMenuItem;
49 import javax.swing.JOptionPane;
50 import javax.swing.JPopupMenu;
51 import javax.swing.JTextField;
52
53 import util.ColorTheme;
54 import util.IconUtil;
55 import util.UtilPrompt;
56
57 public class DataFlowPanel extends TabbedPane {
58     public static interface EventReceiver {
59         boolean addParameterIfAllowed(DataPartParameter param);
60         boolean addResultIfAllowed(DataPartResult result);
61         boolean removeParameterIfAllowed(DataPartParameter param);
62         boolean removeResultIfAllowed(DataPartResult result);
63     }
```

```
64
65     private final static Icon dataFlowIcon = IconUtil.createImageIcon("icons/letter-d-16.png");
66     private static final long serialVersionUID = 1L;
67     //private JDesktopPane theDesktopPane;
68     private ArrayList<DataPart> components = new ArrayList<DataPart>();
69     private ArrayList<DataPart> tempComponents = new ArrayList<DataPart>();
70     private LineConnectPart.Temporary dragLine = null;
71
72     private EventReceiver eventReceiver;
73
74     private Point mouseDraggedComponentStart = new Point();
75     private ArrayList<DataPart> mouseDraggedComponents = new ArrayList<DataPart>();
76     private DataPart mouseOveredComponent;
77     private DataPart.Arrow lineStart;
78     private JPopupMenu popupMenu;
79     private JMenuItem buttonDelete;
80     private LineConnectPart lineHighlight = null;
81     private LineConnectPart lineFocus = null;
82     private DataPart.Arrow arrowFocus = null;
83     private Rectangle dragRectangle;
84     private boolean realMouseDrag = false;
85
86     private JTextField titleTextField = new JTextField();
87
88     private boolean canDataFlowFail = false;
89
90     private int nextSequenceNumber = 0;
91
92     private int allowTypeChangeCount;
93
94     public DataFlowPanel(MainWindow mainw, String title, TabbedPane parent)
95         throws InvalidTitleException {
96         this(mainw, title, parent, new EventReceiver() {
97             @Override
98             public boolean removeResultIfAllowed(DataPartResult result) {
99                 return true;
100            }
101            @Override
102            public boolean removeParameterIfAllowed(DataPartParameter param) {
103                return true;
104            }
105            @Override
106            public boolean addResultIfAllowed(DataPartResult result) {
107                return true;
108            }
109            @Override
110            public boolean addParameterIfAllowed(DataPartParameter param) {
111                return true;
112            }
113        });
114    }
115
116    List<DataPart> getDataParts() {
117        return components;
118    }
```

```
119
120     public void addTempComponent(DataPart p) {
121         tempComponents.add(p);
122     }
123
124     @Override
125     public void applyMouseDrag(Point p1) {
126         Point point = inverseTransformPoint(p1);
127         //draw line from input and output
128
129         if (mouseOveredComponent != null) {
130             //Rectangle r1 = mouseOveredComponent.getBoundingRectangleWithLines();
131             mouseOveredComponent.setDrawFocusBorder(false);
132             mouseOveredComponent = null;
133             //repaint(r1);
134         }
135         if (dragLine != null) {
136             if (arrowFocus != null) {
137                 //Rectangle r1 = arrowFocus.getBoundingOvalRect();
138                 arrowFocus = null;
139                 //repaint(r1);
140             }
141
142             for(int i = components.size()-1; i >= 0; i--) {
143                 DataPart dp = components.get(i);
144                 DataPart.OutputArrow outputArrow = dp.getOutputArrow(point);
145                 if (outputArrow != null) {
146                     arrowFocus = outputArrow;
147                     //Rectangle r = arrowFocus.getBoundingOvalRect();
148                     //repaint(r);
149                     break;
150                 }
151                 DataPart.InputArrow inputArrow = dp.getInputArrow(point);
152                 if (inputArrow != null) {
153                     arrowFocus = inputArrow;
154                     //Rectangle r = arrowFocus.getBoundingOvalRect();
155                     //repaint(r);
156                     break;
157                 }
158             }
159
160             dragLine.setEndPoint(point);
161             //Rectangle r1 = dragLine.setEndPoint(e.getPoint());
162             //repaint(r1);
163             //repaint(dragLine.getBoundingRectangle());
164
165         }
166         for (DataPart dp : components) {
167             if (dp.isMouseInBound(point)){
168                 mouseOveredComponent = dp;
169                 dp.setDrawFocusBorder(true);
170                 break;
171             }
172         }
173         if (realMouseDrag && mouseDraggedComponents.size() > 0) {
```

```
174         //Rectangle r1 = mouseDraggedComponent.getBoundingBoxWithLines();
175         Point p = new Point(point.x-mouseDraggedComponentStart.x, point.y-
176             mouseDraggedComponentStart.y);
177         mouseDraggedComponentStart.x += p.x;
178         mouseDraggedComponentStart.y += p.y;
179         for (DataPart part : mouseDraggedComponents) {
180             part.movePointBox(p);
181         }
182         //Rectangle r2 = mouseDraggedComponent.getBoundingBoxWithLines();
183         //repaint(r1);
184         //repaint(r2);
185     }
186     if (dragRectangle != null) {
187         Point p = new Point(point.x-mouseDraggedComponentStart.x, point.y-
188             mouseDraggedComponentStart.y);
189         mouseDraggedComponentStart.x += p.x;
190         mouseDraggedComponentStart.y += p.y;
191         if (dragRectangle != null) {
192             dragRectangle.width += p.x;
193             dragRectangle.height += p.y;
194         }
195     }
196     public DataFlowPanel(MainWindow mainw, String title, TabbedPane parent, EventReceiver
197     eventReceiver)
198         throws InvalidTitleException {
199         super(mainw, title, parent);
200         this.eventReceiver = eventReceiver;
201         scrollPane.setAutoscrolls(true);
202         UtilPrompt.setPrompt("Dataflow title", titleTextField);
203
204         //popupMenu to delete component
205         popupMenu = new JPopupMenu();
206         buttonDelete = new JMenuItem("Delete");
207         popupMenu.add(buttonDelete);
208         buttonDelete.addActionListener(new ActionListener() {
209             @Override
210             public void actionPerformed(ActionEvent e) {
211                 deleteEvent();
212             }
213         });
214     }
215
216     addMouseMotionListener(new MouseMotionListener(){
217         @Override
218         public void mouseDragged(MouseEvent e) {
219             if (isOverSideOval(e1.getPoint())) {
220                 repaint();
221                 return;
222             }
223             applyMouseDrag(e1.getPoint());
224             repaint();
225         }
226     });
227 }
```

```
226     @Override
227     public void mouseMoved(MouseEvent e1) {
228         if (isOverSideOval(e1.getPoint())) {
229             repaint();
230             return;
231         }
232         MouseEvent e = inverseTransformMouseEvent(e1);
233         Point p = new Point(e.getX(), e.getY());
234
235         if (mouseOveredComponent != null) {
236             //Rectangle r1 = mouseOveredComponent.getBoundingRectangleWithLines();
237             mouseOveredComponent.setDrawFocusBorder(false);
238             mouseOveredComponent = null;
239             //repaint(r1);
240         }
241         if (arrowFocus != null) {
242             //Rectangle r1 = arrowFocus.getBoundingOvalRect();
243             arrowFocus = null;
244             //repaint(r1);
245         }
246         if (lineFocus != null) {
247             //Rectangle r1 = lineFocus.getBoundingRect();
248             lineFocus = null;
249             //repaint(r1);
250         }
251
252         boolean foundComp = false;
253         //Rectangle r = null;
254         for(int i = components.size()-1; i >= 0; i--) {
255             DataPart dp = components.get(i);
256
257             DataPart.OutputArrow outputArrow = dp.getOutputArrow(e.getPoint());
258             if (outputArrow != null) {
259                 arrowFocus = outputArrow;
260                 //r = arrowFocus.getBoundingOvalRect();
261                 foundComp = true;
262                 break;
263             }
264             DataPart.InputArrow inputArrow = dp.getInputArrow(e.getPoint());
265             if (inputArrow != null) {
266                 arrowFocus = inputArrow;
267                 //r = arrowFocus.getBoundingOvalRect();
268                 foundComp = true;
269                 break;
270             }
271
272             if (dp.isMouseInBound(p)){
273                 mouseOveredComponent = dp;
274                 dp.setDrawFocusBorder(true);
275                 //r = dp.getBoundingRectangleWithLines();
276                 foundComp = true;
277                 break;
278             }
279         }
280     }
```

```

281     if (!foundComp) {
282         lineFocus = null;
283         for(int i = components.size()-1; i >= 0; i--) {
284             DataPart dp = components.get(i);
285             lineFocus = getLineIntersecting(dp.getOutputLines(), p);
286             if (lineFocus != null) {
287                 //r = lineFocus.getBoundingRect();
288                 break;
289             }
290         }
291     }
292
293     repaint();
294 }
295 });
296 addMouseListener(new MouseAdapter() {
297     public void mouseReleased(MouseEvent e1){
298         MouseEvent e = inverseTransformMouseEvent(e1);
299
300         realMouseDrag = false;
301
302         //draw line between components
303         if (dragLine != null) {
304             for (DataPart dp : components) {
305                 boolean found = false;
306                 DataPart.OutputArrow outputArrow = dp.getOutputArrow(e.getPoint());
307                 DataPart.InputArrow inputArrow = dp.getInputArrow(e.getPoint());
308                 if (outputArrow != null) {
309                     found = true;
310                     if (lineStart instanceof DataPart.InputArrow) {
311                         inputArrow = (DataPart.InputArrow) lineStart;
312                     }else if (lineStart != outputArrow){
313                         JOptionPane.showMessageDialog(DataFlowPanel.this, "Connection from
Output to Output", "Connection not allowed", JOptionPane.ERROR_MESSAGE);
314                     }
315                 }
316                 if (inputArrow != null && outputArrow == null) {
317                     found = true;
318                     if (lineStart instanceof DataPart.OutputArrow) {
319                         outputArrow = (DataPart.OutputArrow) lineStart;
320                     }else if (lineStart != inputArrow){
321                         JOptionPane.showMessageDialog(DataFlowPanel.this, "Connection from
Input to Input", "Connection not allowed", JOptionPane.ERROR_MESSAGE);
322                     }
323                 }
324                 if (outputArrow != null && inputArrow != null) {
325                     switch (GraphUtil.connectArrows(inputArrow, outputArrow)) {
326                         case SUCCESS:
327                             TypeConnection conn = new TypeConnection(outputArrow.getType(),
inputArrow.getType(), true);
328                             inputArrow.setTypeConnection(conn);
329                             TypeConnectionDialog.show(conn, mainWindow);
330                             if (!conn.isConnected())
331                                 GraphUtil.disconnectArrow(inputArrow, outputArrow);
332                             DataFlowPanel.this.repaint();
333                         }
334                     }
335                 }
336             }
337         }
338     }
339 }
340 
```

```
333                     break;
334         case CYCLE:
335             JOptionPane.showMessageDialog(DataFlowPanel.this, "Cycle in
336             dataflow", "Connection not allowed", JOptionPane.ERROR_MESSAGE);
337         case MULTI_INPUT:
338             JOptionPane.showMessageDialog(DataFlowPanel.this, "Multiple input
339             connections", "Connection not allowed", JOptionPane.ERROR_MESSAGE);
340             break;
341         }
342     }
343     if (found) {
344         break;
345     }
346 }
347 dragLine = null;
348 }
349
350 if (dragRectangle != null) {
351     Rectangle r = getDrawRectangle(dragRectangle);
352     for (DataPart dp : components) {
353         if (dp.isInsideOrIntersects(r)) {
354             mouseDraggedComponents.add(dp);
355         }
356     }
357     for (DataPart dp : mouseDraggedComponents) {
358         dp.setDrawHighlightBorder(true);
359         components.remove(dp);
360         components.add(dp);
361     }
362     dragRectangle = null;
363 }
364
365 repaint();
366 }
367
368 public void mousePressed(MouseEvent e1){
369     if (isOverSideOval(e1.getPoint())) {
370         repaint();
371         return;
372     }
373     MouseEvent e = inverseTransformMouseEvent(e1);
374     Point point = e.getPoint();
375
376     realMouseDrag = true;
377
378     mouseDraggedComponentStart.x = e.getX();
379     mouseDraggedComponentStart.y = e.getY();
380     boolean compIsSelected = false;
381     DataPart selectedComp = null;
382     lineHighlight = null;
383     dragLine = null;
384
385     boolean clearDragComponents = false;
```

```
386
387     for(int i = components.size()-1; i >= 0; i--){
388         DataPart dp = components.get(i);
389         DataPart.OutputArrow outputArrow = dp.getOutputArrow(e.getPoint());
390         if (outputArrow != null) {
391             Point center = outputArrow.getOvalCenter();
392             dragLine = LineConnectPart.makeTemporary(center, center);
393             lineStart = outputArrow;
394             clearDragComponents = true;
395             break;
396         }
397         DataPart.InputArrow inputArrow = dp.getInputArrow(e.getPoint());
398         if (inputArrow != null) {
399             Point center = inputArrow.getOvalCenter();
400             dragLine = LineConnectPart.makeTemporary(center, center);
401             lineStart = inputArrow;
402             clearDragComponents = true;
403             break;
404         }
405         if (!clearDragComponents && dp.isMouseInBound(point) &&
406             !mouseDraggedComponents.contains(dp)) {
407             clearDragComponents = true;
408         }
409     }
410     if (!clearDragComponents) {
411         clearDragComponents = true;
412         for (DataPart dp : mouseDraggedComponents) {
413             if (dp.isMouseInBound(point)) {
414                 clearDragComponents = false;
415                 break;
416             }
417         }
418     }
419
420     if (clearDragComponents && mouseDraggedComponents.size() > 0) {
421         mouseDraggedComponents.clear();
422     }
423
424     if (dragLine == null) {
425         boolean popupSelected = false;
426         for(int i = components.size()-1; i >= 0; i--){
427             DataPart dp = components.get(i);
428             dp.setDrawHighlightBorder(false);
429             if(!compIsSelected && dp.isMouseInBound(mouseDraggedComponentStart)) {
430                 if (clearDragComponents)
431                     mouseDraggedComponents.add(dp);
432                 selectedComp = dp;
433                 compIsSelected = true;
434             }
435             if (!popupSelected) {
436                 //show PopupMenu to delete component
437                 if(e.isPopupTrigger() && dp.isMouseInBound(mouseDraggedComponentStart)
438             ) {
439                 popupSelected = true;
440                 Point p = transformPoint(getTransform(), e.getPoint());
```

```
440                     popupMenu.show(e.getComponent(), p.x, p.y);
441                 }
442             //show PopupMenu to delete line
443             if(e.isPopupTrigger() && lineFocus != null){
444                 popupSelected = true;
445                 Point p = transformPoint(getTransform(), e.getPoint());
446                 popupMenu.show(e.getComponent(), p.x, p.y);
447             }
448         }
449     }
450 }
451 for (DataPart dp : mouseDraggedComponents) {
452     dp.setDrawHighlightBorder(true);
453 }
454 if (compIsSelected) {
455     components.remove(mouseDraggedComponents.get(0));
456     components.add(mouseDraggedComponents.get(0));
457 } else if (dragLine == null) {
458     for(int i = components.size()-1; i >= 0; i--){
459         DataPart dp = components.get(i);
460         lineHighlight = getLineIntersecting(dp.getOutputLines(), e.getPoint());
461         if (lineHighlight != null)
462             break;
463     }
464     if (!e.isPopupTrigger() && lineHighlight != null && e.getClickCount() == 2) {
465         TypeConnection conn = lineHighlight.getInputArrow().getTypeConnection();
466         TypeConnectionDialog.show(conn, mainWindow);
467         if (!conn.isConnected()) {
468             GraphUtil.disconnectArrow(lineHighlight.getInputArrow(),
469             lineHighlight.getOutputArrow());
470             lineHighlight = null;
471             lineFocus = null;
472         }
473     }
474 }
475
476 if (mouseDraggedComponents.isEmpty() && arrowFocus == null &&
477     lineHighlight == null && dragLine == null) {
478     dragRectangle = new Rectangle(point.x, point.y, 0, 0);
479 }
480
481 if (!e.isPopupTrigger() && e.getClickCount() == 2) {
482     dragRectangle = null;
483     dragLine = null;
484     if (compIsSelected) {
485         selectedComp.doubleClick(mainWindow, mainWindow);
486     } else if (arrowFocus != null) {
487         if (arrowFocus instanceof DataPart.InputArrow)
488             ShowTypePanel.showTypeDialog(arrowFocus.getType(), mainWindow, false);
489         else
490             ShowTypePanel.showTypeDialog(arrowFocus.getType(), mainWindow, true);
491         arrowFocus = null;
492     }
493 }
```

```
495             repaint();
496         }
497     });
498 }
499 }
500
501 private void deleteDataPart(DataPart dp) {
502     if (dp.markDeletedIfAllowed()) {
503         if (dp instanceof DataPartParameter &&
504             !eventReceiver.removeParameterIfAllowed((DataPartParameter)dp)) {
505             return;
506         } else if (dp instanceof DataPartResult &&
507             !eventReceiver.removeResultIfAllowed((DataPartResult)dp)) {
508             return;
509         }
510         GraphUtil.disconnectAllArrows(dp);
511         components.remove(dp);
512     }
513     resetCanDataFlowFail();
514 }
515
516 //delete line or delete component
517 private void deleteEvent() {
518     if (mouseDraggedComponents.isEmpty()) {
519         DataPart next = components.size() > 0 ? components.get(0) : null;
520         for (int i = 0; i < components.size(); i++) {
521             DataPart dp = next;
522             if (i < components.size()-1) {
523                 next = components.get(i+1);
524             }
525             if (dp.isHighlighted()) {
526                 deleteDataPart(dp);
527             }
528         }
529         if (next.isHighlighted())
530             deleteDataPart(next);
531     } else {
532         for (DataPart dp : mouseDraggedComponents) {
533             deleteDataPart(dp);
534         }
535         mouseDraggedComponents.clear();
536     }
537     if(lineHighlight != null){
538         if (lineHighlight.sameLine(lineFocus))
539             lineFocus = null;
540         GraphUtil.disconnectArrow(lineHighlight.getInputArrow(), lineHighlight.getOutputArrow
541             ());
541         lineHighlight = null;
542     }
543     repaint();
544 }
545
546 public LineConnectPart getLineIntersecting(ArrayList<LineConnectPart> lines, Point p){
547     for(LineConnectPart line: lines){
548         if(line.intersects(p))
```

```
549             return line;
550     }
551     return null;
552 }
553
554 public boolean oneLineIntersect(ArrayList<LineConnectPart> lines, Point p){
555     return getLineIntersecting(lines, p) != null;
556 }
557
558 @Override
559 public void paintComponent(Graphics g) {
560     Graphics2D g2 = (Graphics2D) g;
561     super.paintComponent(g);
562     g2.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING, RenderingHints.
563     VALUE_TEXT_ANTIALIAS_ON);
564     g2.transform(getTransform());
565
566     setBackground(Color.WHITE);
567     for (DataPart c : components) {
568         c.paintOutputLines(g);
569     }
570     if (lineFocus != null) {
571         Stroke s = g2.getStroke();
572         g2.setStroke(new BasicStroke(6));
573         g2.setColor(ColorTheme.HOVERED);
574         g2.draw(lineFocus.getLine());
575         g2.setColor(Color.black);
576         g2.setStroke(s);
577     }
578     if (lineHighlight != null) {
579         Stroke s = g2.getStroke();
580         g2.setStroke(new BasicStroke(4));
581         g2.setColor(ColorTheme.SELECTED);
582         g2.draw(lineHighlight.getLine());
583         g2.setColor(Color.black);
584         g2.setStroke(s);
585     }
586     for (DataPart c : components) {
587         c.paint(g);
588     }
589     if (arrowFocus != null) {
590         Point p = arrowFocus.getOvalCenter();
591         int d = arrowFocus.getDiameter();
592         g2.setColor(ColorTheme.HOVERED);
593         g2.fillOval(p.x-3-d/2, p.y-3-d/2, d+6, d+6);
594         g2.setColor(Color.black);
595     }
596     g2.setStroke(new BasicStroke(2));
597     if (dragLine != null) {
598         dragLine.paint(g2);
599     }
600     if (dragRectangle != null) {
601         Rectangle r = getDrawRectangle(dragRectangle);
602         Color c = g2.getColor();
603         g2.setStroke(new BasicStroke(1));
```

```
603         g2.setColor(ColorTheme.TRANS_LIGHT_GRAY);
604         g2.fill(r);
605         g2.setColor(ColorTheme.NONTRANS_LIGHT_GRAY);
606         g2.draw(r);
607         g2.setColor(c);
608     }
609
610     drawBorder(g);
611 }
612
613 public void validateAddDataPartResult(String newName) throws InvalidDataPartException {
614     for (DataPart p : components) {
615         if (!(p instanceof DataPartResult))
616             continue;
617         DataPartResult res = (DataPartResult)p;
618         if (res.getName().equals(newName)) {
619             throw new InvalidDataPartException("Output '"+newName+"' already exists");
620         }
621     }
622 }
623
624 public void validateAddDataPartParameter(String newName) throws InvalidDataPartException {
625     for (DataPart p : components) {
626         if (!(p instanceof DataPartParameter))
627             continue;
628         DataPartParameter res = (DataPartParameter)p;
629         if (res.getName().equals(newName)) {
630             throw new InvalidDataPartException("Input '"+newName+"' already exists");
631         }
632     }
633 }
634
635 private void addDataPart(DataPart part) {
636     addDataPart(part, true);
637 }
638
639 public void addDataPart(DataPart part, boolean setLocation) {
640     if (part instanceof DataPartParameter) {
641         if (!eventReceiver.addParameterIfAllowed((DataPartParameter)part))
642             return;
643     } else if (part instanceof DataPartResult) {
644         if (!eventReceiver.addResultIfAllowed((DataPartResult)part))
645             return;
646     }
647     if (setLocation) {
648         Rectangle screen = scrollPane.getViewport().getViewRect();
649         Point screenPoint = inverseTransformPoint(screen.getLocation());
650
651         Point prev = part.getPointBoxLocation();
652         int w = inverseZoom(screen.width/2.0);
653         int h = inverseZoom(screen.height/2.0);
654         part.movePointBox(new Point(screenPoint.x + w - prev.x, screenPoint.y + h - prev.y));
655     }
656     part.setSequenceNumber(nextSequenceNumber++);
657     components.add(part);
```

```
658         resetCanDataFlowFail();
659         repaint();
660     }
661
662     void setDataPartsClearTemps(List<DataPart> parts) {
663         tempComponents.clear();
664         components = new ArrayList<>(parts);
665     }
666
667     public DataPartRecordConstructor getNewDataPartRecordConstructor(TypeRecord type) {
668         return new DataPartRecordConstructor(type, this, mainWindow.getGraphics());
669     }
670
671     public DataPartRecordConstructor addNewDataPartRecordConstructor(TypeRecord type) {
672         DataPartRecordConstructor ret = getNewDataPartRecordConstructor(type);
673         addDataPart(ret);
674         return ret;
675     }
676
677     public DataPartSink getNewDataPartSink(Type type) {
678         return new DataPartSink(type, this, mainWindow.getGraphics());
679     }
680
681     public DataPartSink addNewDataPartSink(Type type) {
682         DataPartSink ret = getNewDataPartSink(type);
683         addDataPart(ret);
684         return ret;
685     }
686
687     public DataPartConstant getNewDataPartConstant(Constant val) {
688         DataPartConstant p = new DataPartConstant(val, this, mainWindow.getGraphics());
689         return p;
690     }
691
692     public DataPartConstant addNewDataPartConstant(Constant val) {
693         DataPartConstant ret = getNewDataPartConstant(val);
694         addDataPart(ret);
695         return ret;
696     }
697
698     public DataPartArithmetic getNewDataPartArithmetic(DataPartArithmetic.ArithmeticType type) {
699         DataPartArithmetic p = new DataPartArithmetic(type, this, mainWindow.getGraphics());
700         return p;
701     }
702
703     public DataPartArithmetic addNewDataPartArithmetic(DataPartArithmetic.ArithmeticType type) {
704         DataPartArithmetic p = getNewDataPartArithmetic(type);
705         addDataPart(p);
706         return p;
707     }
708
709     public DataPartStringOperation getNewDataPartStringOperation(DataPartStringOperation.
710         StringOperationType type) {
711         DataPartStringOperation p = new DataPartStringOperation(type, this, mainWindow.getGraphics()
712             ());
713     }
```

```
711         return p;
712     }
713
714     public DataPartStringOperation addNewDataPartStringOperation(DataPartStringOperation.
715         StringOperationType type) {
715         DataPartStringOperation p = getNewDataPartStringOperation(type);
716         addDataPart(p);
717         return p;
718     }
719
720     public DataPartStringFormat getNewDataPartStringFormat(String fmt) throws
721         InvalidDataPartException {
721         DataPartStringFormat p = new DataPartStringFormat(fmt, this, mainWindow.getGraphics());
722         return p;
723     }
724
725     public DataPartStringFormat addNewDataPartStringFormat(String fmt) throws
726         InvalidDataPartException {
726         DataPartStringFormat p = getNewDataPartStringFormat(fmt);
727         addDataPart(p);
728         return p;
729     }
730
731     public DataPartComparison getNewDataPartComparison(DataPartComparison.ComparisonType compType,
732         Type origType, Type dataType)
732         throws InvalidDataPartException {
733         DataPartComparison p = new DataPartComparison(compType, origType, dataType, this,
733             mainWindow.getGraphics());
734         return p;
735     }
736
737     public DataPartComparison addNewDataPartComparison(DataPartComparison.ComparisonType compType,
738         Type origType, Type dataType)
738         throws InvalidDataPartException {
739         DataPartComparison p = getNewDataPartComparison(compType, origType, dataType);
740         addDataPart(p);
741         return p;
742     }
743
744     public DataPartCast getNewDataPartCast(Type origFrom, Type from, Type to) throws
745         InvalidDataPartException {
745         DataPartCast p = new DataPartCast(origFrom, from, to, this, mainWindow.getGraphics());
746         return p;
747     }
748
749     public DataPartCast addNewDataPartCast(Type origFrom, Type from, Type to) throws
750         InvalidDataPartException {
750         DataPartCast p = getNewDataPartCast(origFrom, from, to);
751         addDataPart(p);
752         return p;
753     }
754
755     public DataPartFunctional getNewDataPartFunctional(DataPartFunctional.FunctionalType type,
756         String subDataFlowTitle,
756             List<RecordColumn> inputs, List<RecordColumn> outputs)
```

```
757         throws InvalidTitleException, InvalidDataPartException {
758     DataPartFunctional p = new DataPartFunctional(type, mainWindow, subDataFlowTitle, this,
759         mainWindow.getGraphics(),
760         inputs, outputs);
761     validateAddISubDataFlow(p);
762     return p;
763 }
764 
765     public DataPartFunctional getNewDataPartFunctional(DataPartFunctional.FunctionalType type,
766             String subDataFlowTitle)
767         throws InvalidTitleException, InvalidDataPartException {
768     DataPartFunctional p = new DataPartFunctional(type, mainWindow, subDataFlowTitle, this,
769         mainWindow.getGraphics());
770     validateAddISubDataFlow(p);
771     return p;
772 }
773 
774     public DataPartFunctional addNewDataPartFunctional(DataPartFunctional.FunctionalType type,
775             String subDataFlowTitle)
776         throws InvalidTitleException, InvalidDataPartException {
777     DataPartFunctional p = getNewDataPartFunctional(type, subDataFlowTitle);
778     addDataPart(p);
779     return p;
780 }
781 
782 
783     public DataPartListOperation getNewDataPartListOperation(DataPartListOperation.
784             ListOperationType type) {
785         DataPartListOperation p = new DataPartListOperation(type, this, mainWindow.getGraphics());
786         return p;
787 }
788 
789     public DataPartListOperation addNewDataPartListOperation(DataPartListOperation.
790             ListOperationType type) {
791         DataPartListOperation p = getNewDataPartListOperation(type);
792         addDataPart(p);
793         return p;
794 }
795 
796     public DataPartListOperation getNewDataPartListOperation(DataPartListOperation.
797             ListOperationType type, List<Type> origOpTypes, List<Type> opTypes) {
798         DataPartListOperation p = new DataPartListOperation(type, origOpTypes, opTypes, this,
799             mainWindow.getGraphics());
800         return p;
801 }
802 
803     public DataPartParameter getNewDataPartParameter(String name, Type origType, Type type,
804             Predicate<DataPartParameter> deleteEvent, boolean allowChangeType) throws
805             InvalidDataPartException {
```

```
802     DataPartParameter p = new DataPartParameter(origType, type, name, this, mainWindow.  
803         getGraphics(), deleteEvent, allowChangeType);  
804     validateAddDataPartParameter(p.getName());  
805     return p;  
806 }  
807  
808 public DataPartParameter addNewDataPartParameter(String name, Type origType, Type type,  
809         Predicate<DataPartParameter> deleteEvent, boolean allowChangeType) throws  
810     InvalidDataPartException {  
811     DataPartParameter p = getNewDataPartParameter(name, origType, type, deleteEvent,  
812         allowChangeType);  
813     addDataPart(p);  
814     return p;  
815 }  
816  
817 public DataPartParameter addNewDataPartParameter(String name, Type origType, Type type,  
818         Predicate<DataPartParameter> deleteEvent) throws InvalidDataPartException {  
819     DataPartParameter p = getNewDataPartParameter(name, origType, type, deleteEvent, true);  
820     addDataPart(p);  
821     return p;  
822 }  
823  
824 public DataPartResult getNewDataPartResult(String name, Type origType, Type type,  
825         Predicate<DataPartResult> deleteEvent, boolean allowChangeType) throws  
826     InvalidDataPartException {  
827     DataPartResult r = new DataPartResult(origType, type, name, this, mainWindow.getGraphics()  
828         , deleteEvent, allowChangeType);  
829     validateAddDataPartResult(r.getName());  
830     if (getParentTabbedPane() instanceof ControlFlowPanel)  
831         r.setInputArrowListeners(Arrays.asList(r));  
832     return r;  
833 }  
834  
835 public DataPartResult addNewDataPartResult(String name, Type origType, Type type,  
836         Predicate<DataPartResult> deleteEvent, boolean allowChangeType) throws  
837     InvalidDataPartException {  
838     DataPartResult r = getNewDataPartResult(name, origType, type, deleteEvent, allowChangeType  
839         );  
840     addDataPart(r);  
841     return r;  
842 }  
843  
844 private void validateAddISubDataFlow(IDataPartSubFlow subFlow) throws InvalidTitleException{  
845     String subTitle = subFlow.getSubDataFlowPanel().getTitle();  
846     for(DataPart dp: components){  
847         if (!(dp instanceof IDataPartSubFlow))  
848             continue;  
849         IDataPartSubFlow res = (IDataPartSubFlow)dp;
```

```
850         if (res.getSubDataFlowPanel().getTitle().equals(subTitle)) {
851             mainWindow.removeExplorerItem(subFlow.getSubDataFlowPanel());
852             throw new InvalidTitleException("Subdataflow '"+subTitle+"' already exists");
853         }
854     }
855 }
856
857 public DataPartSubFlow getNewDataPartSubFlow(String title)
858     throws InvalidDataPartException, InvalidTitleException{
859     DataPartSubFlow subFlow = new DataPartSubFlow(title, this, mainWindow, mainWindow.
860     getGraphics());
861     validateAddISubDataFlow(subFlow);
862     return subFlow;
863 }
864
865 public DataPartSubFlow addNewDataPartSubFlow(String title)
866     throws InvalidDataPartException, InvalidTitleException{
867     DataPartSubFlow subFlow = getNewDataPartSubFlow(title);
868     addDataPart(subFlow);
869     return subFlow;
870 }
871
872 @Override
873 public Icon getIcon() {
874     return dataFlowIcon;
875 }
876
877 private void resetCanDataFlowFail() {
878     boolean can = false;
879     for (DataPart dp : components) {
880         if (dp.canFail()) {
881             can = true;
882         }
883     }
884     if (can != canDataFlowFail) {
885         canDataFlowFail = can;
886         getParentTabbedPane().childFailStateChanged();
887     } else {
888         canDataFlowFail = can;
889     }
890 }
891
892 public boolean canFail() {
893     return canDataFlowFail;
894 }
895
896 @Override
897 public void childFailStateChanged() {
898     resetCanDataFlowFail();
899 }
900
901 public List<RecordColumn> getArguments() {
902     ArrayList<DataPartParameter> args = new ArrayList<>();
903     for (DataPart dp : components) {
904         if (dp instanceof DataPartParameter) {
```

```
904             DataPartParameter arg = (DataPartParameter)dp;
905             args.add(arg);
906         }
907     }
908     Collections.sort(args, (x,y) -> {
909         int xn = x.getSequenceNumber();
910         int yn = y.getSequenceNumber();
911         if (xn < yn) return -1; else if (xn == yn) return 0; else return 1;
912     });
913     ArrayList<RecordColumn> ret = new ArrayList<>();
914     for (DataPartParameter p : args) {
915         ret.add(new RecordColumn(p.getName(), p.getType()));
916     }
917     return ret;
918 }
919
920     public DataPartParameter getDataPartParameter(String name) {
921         for (DataPart dp : components) {
922             if (dp instanceof DataPartParameter) {
923                 DataPartParameter arg = (DataPartParameter)dp;
924                 if (arg.getName().equals(name))
925                     return arg;
926             }
927         }
928         throw new IllegalArgumentException("No such parameter data flow " + name);
929     }
930
931     public DataPartResult getDataPartResult(String name) {
932         for (DataPart dp : components) {
933             if (dp instanceof DataPartResult) {
934                 DataPartResult arg = (DataPartResult)dp;
935                 if (arg.getName().equals(name))
936                     return arg;
937             }
938         }
939         throw new IllegalArgumentException("No such result data flow " + name);
940     }
941
942     public List<RecordColumn> getResults() {
943         ArrayList<DataPartResult> args = new ArrayList<>();
944         for (DataPart dp : components) {
945             if (dp instanceof DataPartResult) {
946                 DataPartResult arg = (DataPartResult)dp;
947                 args.add(arg);
948             }
949         }
950         Collections.sort(args, (x,y) -> {
951             int xn = x.getSequenceNumber();
952             int yn = y.getSequenceNumber();
953             if (xn < yn) return -1; else if (xn == yn) return 0; else return 1;
954         });
955         ArrayList<RecordColumn> ret = new ArrayList<>();
956         for (DataPartResult p : args) {
957             ret.add(new RecordColumn(p.getName(), p.getType()));
958         }
```

```
959         return ret;
960     }
961
962     @Override
963     public Predicate<DataPartParameter> getChildParamaterDeleteEvent() {
964         return (param) -> {
965             DataFlowPanel paramPanel = param.getDataFlowPanel();
966             boolean accepted = false;
967             for (DataPart p : components) {
968                 if (p instanceof DataPartSubFlow) {
969                     DataPartSubFlow sub = (DataPartSubFlow)p;
970                     if (!accepted && paramPanel == sub.getSubDataFlowPanel() &&
971                         sub.isParameterTypeConnected(param.getName())) {
972                         int confirm = JOptionPane.showOptionDialog(mainWindow,
973                             "Delete will affect dataflow '"+getTitle()+"'.\nDo you want to
974                             delete?", 
975                             "Delete confirmation", JOptionPane.YES_NO_OPTION,
976                             JOptionPane.QUESTION_MESSAGE, null, null);
977                         if (confirm != 0)
978                             return false;
979                         accepted = true;
980                     }
981                 }
982             }
983             return true;
984         };
985
986     @Override
987     public Predicate<DataPartResult> getChildResultDeleteEvent() {
988         return (result) -> {
989             DataFlowPanel paramPanel = result.getDataFlowPanel();
990             boolean accepted = false;
991             for (DataPart p : components) {
992                 if (p instanceof DataPartSubFlow) {
993                     DataPartSubFlow sub = (DataPartSubFlow)p;
994                     if (!accepted && paramPanel == sub.getSubDataFlowPanel() &&
995                         sub.isResultTypeConnected(result.getName())) {
996                         int confirm = JOptionPane.showOptionDialog(mainWindow,
997                             "Delete will affect dataflow '"+getTitle()+"'.\nDo you want to
998                             delete?", 
999                             "Delete confirmation", JOptionPane.YES_NO_OPTION,
1000                            JOptionPane.QUESTION_MESSAGE, null, null);
1001                         if (confirm != 0)
1002                             return false;
1003                         accepted = true;
1004                     }
1005                 }
1006             }
1007             return true;
1008         };
1009
1010     @Override
1011     public Dimension getMinimumDimension() {
```

```
1012     Dimension min = new Dimension(0,0);
1013     for (DataPart dp : components) {
1014         Rectangle r = dp.getBoundingRectangleWithLines();
1015         min.width = Math.max(r.x+r.width, min.width);
1016         min.height = Math.max(r.y+r.height, min.height);
1017     }
1018     min.width += 50;
1019     min.height += 50;
1020     return min;
1021 }
1022
1023 @Override
1024 public boolean hasDataFlowWithTitle(String title) {
1025     for (DataPart p : components) {
1026         if (p instanceof DataPartSubFlow &&
1027             ((DataPartSubFlow) p).getSubDataFlowPanel().getTitle().equals(title))
1028             return true;
1029     }
1030     return false;
1031 }
1032
1033 @Override
1034 public void validateDuplicateTitle(String title)
1035     throws InvalidTitleException {
1036     if (getParentTabbedPane().hasDataFlowWithTitle(title))
1037         throw new InvalidTitleException("Dataflow title '"+title+"' already exists");
1038 }
1039
1040 private ArrayList<DataPart> getCheckComponents() {
1041     ArrayList<DataPart> comps = components;
1042     if (tempComponents.size() > 0) {
1043         comps = new ArrayList<>(components);
1044         comps.addAll(tempComponents);
1045     }
1046     return comps;
1047 }
1048
1049 @Override
1050 public boolean isParameterConnected(DataFlowPanel containingPanel,
1051                                     DataPartParameter dataPartParameter) {
1052     DataFlowPanel paramPanel = dataPartParameter.getDataFlowPanel();
1053     for (DataPart p : getCheckComponents()) {
1054         if (p instanceof IDataPartSubFlow) {
1055             IDataPartSubFlow sub = (IDataPartSubFlow)p;
1056             if (paramPanel == sub.getSubDataFlowPanel() &&
1057                 p.isParameterTypeConnected(dataPartParameter.getName())) {
1058                 return true;
1059             }
1060         }
1061     }
1062     return false;
1063 }
1064
1065 @Override
1066 public void parameterTypeChanged(DataFlowPanel containingPanel,
```

```
1067         DataPartParameter dataPartParameter) {
1068     boolean disconnect = true;
1069     if (allowTypeChangeCount > 0) {
1070         disconnect = false;
1071         --allowTypeChangeCount;
1072     }
1073     DataFlowPanel paramPanel = dataPartParameter.getDataFlowPanel();
1074     for (DataPart p : getCheckComponents()) {
1075         if (p instanceof IDataPartSubFlow) {
1076             IDataPartSubFlow sub = (IDataPartSubFlow)p;
1077             if (paramPanel == sub.getSubDataFlowPanel()) {
1078                 if (disconnect)
1079                     p.disconnectInputTypeIfConnected(dataPartParameter.getName());
1080                 //System.out.println("NEW TYPE "+dataPartParameter.getType());
1081                 p.changeInputPortType(dataPartParameter.getName(), dataPartParameter.getType())
1082             );
1083         }
1084     }
1085 }
1086
1087 @Override
1088 public void changeParameterName(DataFlowPanel containingPanel,
1089         DataPartParameter dataPartParameter, String oldName) {
1090     DataFlowPanel paramPanel = dataPartParameter.getDataFlowPanel();
1091     for (DataPart p : getCheckComponents()) {
1092         if (p instanceof IDataPartSubFlow) {
1093             IDataPartSubFlow sub = (IDataPartSubFlow)p;
1094             if (paramPanel == sub.getSubDataFlowPanel()) {
1095                 p.changeInputPortName(oldName, dataPartParameter.getName());
1096             }
1097         }
1098     }
1099 }
1100
1101 @Override
1102 public void changeResultName(DataFlowPanel containingPanel,
1103         DataPartResult dataPartResult, String oldName) {
1104     DataFlowPanel resultPanel = dataPartResult.getDataFlowPanel();
1105     for (DataPart p : getCheckComponents()) {
1106         if (p instanceof IDataPartSubFlow) {
1107             IDataPartSubFlow sub = (IDataPartSubFlow)p;
1108             if (resultPanel == sub.getSubDataFlowPanel()) {
1109                 p.changeOutputPortName(oldName, dataPartResult.getName());
1110             }
1111         }
1112     }
1113 }
1114
1115 @Override
1116 public boolean isResultConnected(DataFlowPanel containingPanel,
1117         DataPartResult dataPartResult) {
1118     DataFlowPanel resultPanel = dataPartResult.getDataFlowPanel();
1119     for (DataPart p : getCheckComponents()) {
1120         if (p instanceof IDataPartSubFlow) {
```

```
1121             IDataPartSubFlow sub = (IDataPartSubFlow)p;
1122             if (resultPanel == sub.getSubDataFlowPanel() &&
1123                 p.isResultTypeConnected(dataPartResult.getName())) {
1124                 return true;
1125             }
1126         }
1127     }
1128     return false;
1129 }
1130
1131 @Override
1132 public void resultTypeChanged(DataFlowPanel containingPanel,
1133                               DataPartResult dataPartResult) {
1134     boolean disconnect = true;
1135     if (allowTypeChangeCount > 0) {
1136         disconnect = false;
1137         --allowTypeChangeCount;
1138     }
1139     DataFlowPanel resultPanel = dataPartResult.getDataFlowPanel();
1140     for (DataPart p : getCheckComponents()) {
1141         if (p instanceof IDataPartSubFlow) {
1142             IDataPartSubFlow sub = (IDataPartSubFlow)p;
1143             if (resultPanel == sub.getSubDataFlowPanel()) {
1144                 if (disconnect)
1145                     p.disconnectOutputTypeIfConnected(dataPartResult.getName());
1146                 p.changeOutputPortType(dataPartResult.getName(), dataPartResult.getType());
1147             }
1148         }
1149     }
1150 }
1151
1152 private DataPart getSequenceNumberComponent(int seq) {
1153     for (DataPart p : components) {
1154         if (p.getSequenceNumber() == seq) {
1155             return p;
1156         }
1157     }
1158     return null;
1159 }
1160
1161 @Override
1162 public void scrollToComponent(int seqNumber) {
1163     DataPart p = getSequenceNumberComponent(seqNumber);
1164     if (p == null) {
1165         JOptionPane.showMessageDialog(mainWindow, "Cannot find component",
1166                                   "Unable to scroll", JOptionPane.INFORMATION_MESSAGE);
1167         return;
1168     }
1169     if (mouseDraggedComponents.size() > 0) {
1170         for (DataPart part : mouseDraggedComponents) {
1171             part.setDrawHighlightBorder(false);
1172         }
1173     }
1174     for (DataPart dp : components)
1175         dp.setDrawHighlightBorder(false);
```

```
1176     lineHighlight = null;
1177     mouseDraggedComponents.add(p);
1178     p.setDrawHighlightBorder(true);
1179     scrollToPoint(p.getPointBoxLocation());
1180     repaint();
1181 }
1182
1183 @Override
1184 public void scrollToPath(int startSeq, int evt, int endSeq) {
1185     throw new RuntimeException("This function should never get called!");
1186 }
1187
1188 public void allowTypeChanges(int count) {
1189     allowTypeChangeCount += count;
1190 }
1191 }
```

LISTING E.97: gui/DataFlowPanel.java

```
1 package gui;
2
3 import lang.type.Type;
4
5 public interface TypeChangedListener {
6     void typeChanged(Type newType);
7 }
```

LISTING E.98: gui/TypeChangedListener.java

```
1 package gui;
2
3 import java.awt.Window;
4 import java.awt.event.WindowAdapter;
5 import java.awt.event.WindowEvent;
6
7 import javax.swing.JDialog;
8 import javax.swing.JOptionPane;
9 import javax.swing.JPanel;
10 import javax.swing.JScrollPane;
11
12 import lang.type.TypeConnection;
13
14 @SuppressWarnings("serial")
15 public class TypeConnectionDialog extends JPanel {
16     private TypeConnectionPanel.GlassPanel glassPanel;
17     private JDialog window;
18
19     public TypeConnectionDialog(JDialog window, JScrollPane scrollPane) {
20         glassPanel = new TypeConnectionPanel.GlassPanel(window, scrollPane);
21         this.window = window;
22         window.setGlassPane(glassPanel);
23         glassPanel.setVisible(true);
24     }
25 }
```

```

26     public static void show(TypeConnection conn, Window owner) {
27         JDialog dialog = new JDialog(owner);
28         dialog.setSize(500, 300);
29         dialog.setLocationRelativeTo(owner);
30         dialog.setModal(true);
31         dialog.setDefaultCloseOperation(JDialog.DO_NOTHING_ON_CLOSE);
32
33         TypeConnectionDialog tcd = new TypeConnectionDialog(dialog, null);
34         TypeConnectionPanel.Container panel = tcd.getConnectionPanel(conn);
35         dialog.addWindowListener(new WindowAdapter() {
36             @Override
37             public void windowClosing(WindowEvent e) {
38                 if (panel.get().isConnected()) {
39                     dialog.dispose();
40                     return;
41                 }
42                 int confirm = JOptionPane.showOptionDialog(dialog,
43                     "Not all data inputs have been connected.\n"+"Close and discard connection
44                     ?",
45                     "Exit confirmation", JOptionPane.YES_NO_OPTION,
46                     JOptionPane.QUESTION_MESSAGE, null, null, null);
47                 if (confirm == 0) {
48                     dialog.dispose();
49                 }
50             });
51
52         dialog.getContentPane().add(panel);
53         dialog.setVisible(true);
54     }
55
56     public TypeConnectionPanel.Container getConnectionPanel(TypeConnection conn) {
57         return TypeConnectionPanel.get(conn, window, glassPanel);
58     }
59 }
```

LISTING E.99: gui/TypeConnectionDialog.java

```

1 package gui;
2
3 import java.awt.Dimension;
4 import java.awt.EventQueue;
5 import java.awt.FlowLayout;
6 import java.awt.Graphics;
7 import java.awt.Graphics2D;
8 import java.awt.Point;
9 import java.awt.Polygon;
10 import java.awt.Rectangle;
11 import java.awt.event.MouseAdapter;
12 import java.awt.event.MouseEvent;
13 import java.awt.event.MouseMotionListener;
14 import java.awt.geom.AffineTransform;
15 import java.awt.geom.Ellipse2D;
16 import java.awt.geom.NoninvertibleTransformException;
17 import java.awt.geom.Point2D;
```

```
18 import java.lang.reflect.InvocationTargetException;
19 import java.util.ArrayList;
20 import java.util.concurrent.atomic.AtomicBoolean;
21 import java.util.concurrent.atomic.AtomicInteger;
22 import java.util.function.Predicate;
23
24 import javax.swing.Icon;
25 import javax.swing.JPanel;
26 import javax.swing.JScrollPane;
27 import javax.swing.JViewport;
28 import javax.swing.event.ChangeEvent;
29 import javax.swing.event.ChangeListener;
30
31 import gui.dataflow.DataPartParameter;
32 import gui.dataflow.DataPartResult;
33 import util.ColorTheme;
34 import util.UtilRegex;
35 import util.UtilScrollPane;
36
37 @SuppressWarnings("serial")
38 public abstract class TabbedPane extends JPanel {
39     private final static double zoomInScale = 1.25;
40     private final static double zoomOutScale = 0.8;
41     private final static int scrollStep = 4;
42
43     protected ScrollPane scrollPane = new ScrollPane();
44     protected MainWindow mainWindow;
45     private String tabbedPanelTitle;
46     private ArrayList<TitleChangedListener> titleChangedListeners = new ArrayList<>();
47     private TabbedPane parentTabbedPane;
48     private AffineTransform zoomTransform = new AffineTransform();
49
50     private Point markedSideOvalPoint;
51     private AtomicInteger markedSideOvalIndex = new AtomicInteger(-1);
52     private final static int cornerDia = 48;
53     private final static int halfCorner = cornerDia/2;
54     private final static int sideDia = 36;
55     private final static int halfSide = sideDia/2;
56     private final static int quardCorner = halfCorner/2 - 4;
57     private final static int quardSide = halfSide/2 - 2;
58
59     private MoveThread moveThread;
60
61     private class MoveThread extends Thread {
62         private final static int sleepTime = 10;
63         private AtomicBoolean isStopped = new AtomicBoolean();
64
65         @Override
66         public void run() {
67             while (true) {
68                 try {
69                     if (isStopped.get())
70                         return;
71                     EventQueue.invokeAndWait(() -> moveScreen());
72                     Thread.sleep(sleepTime);
73                 } catch (Exception e) {
74                     e.printStackTrace();
75                 }
76             }
77         }
78     }
79 }
```

```
73             } catch (InterruptedException | InvocationTargetException e) {}
74         }
75     }
76
77     public void markStopped() {
78         isStopped.set(true);
79     }
80
81     public boolean isStarted() {
82         return !isStopped.get();
83     }
84 }
85
86 private void startMoveThread() {
87     if (moveThread != null) {
88         return;
89     }
90     moveThread = new MoveThread();
91     moveThread.start();
92 }
93
94 private void stopMoveThread() {
95     if (moveThread != null) {
96         moveThread.markStopped();
97         moveThread = null;
98     }
99 }
100
101 private class MouseAdapt extends MouseAdapter implements MouseMotionListener {
102     private boolean doStart = false;
103
104     @Override
105     public void mouseExited(MouseEvent e) {
106         unmarkSideOval();
107         doStart = false;
108         stopMoveThread();
109         repaint();
110     }
111     @Override
112     public void mousePressed(MouseEvent e) {
113         TabbedPane.this.requestFocusInWindow();
114         if (getSideOvalIndex(e.getPoint()) != -1) {
115             doStart = true;
116             startMoveThread();
117         }
118     }
119     @Override
120     public void mouseReleased(MouseEvent e) {
121         doStart = false;
122         stopMoveThread();
123     }
124     @Override
125     public void mouseMoved(MouseEvent e) {
126         markSideOval(e.getPoint());
127         if (doStart)
```

```
128         startMoveThread();
129     }
130     @Override
131     public void mouseDragged(MouseEvent e) {
132         doStart = true;
133         markSideOval(e.getPoint());
134         startMoveThread();
135     }
136 }
137
138 public TabbedPane(MainWindow mainWindow, String title, TabbedPane parent)
139     throws InvalidTitleException {
140     super(new FlowLayout());
141
142     if (!UtilRegex.isIdentifier(title))
143         throw new InvalidTitleException("Invalid title '"+title+"'");
144
145     this.mainWindow = mainWindow;
146     tabbedPanelTitle = title;
147     parentTabbedPane = parent;
148
149     scrollPane.setViewport().addChangeListener(new ChangeListener() {
150         @Override
151         public void stateChanged(ChangeEvent e) {
152             resetPreferredSize();
153         }
154     });
155     scrollPane.setViewport().setScrollMode(JViewport.SIMPLE_SCROLL_MODE);
156     this.setAutoscrolls(true);
157
158     MouseAdapt a = new MouseAdapt();
159     addMouseListener(a);
160     addMouseMotionListener(a);
161 }
162
163 public void resetPreferredSize() {
164     if (moveThread != null && moveThread.isStarted())
165         return;
166
167     Dimension min = transformDimension(getTransform(), getMinimumDimension());
168     Dimension view = getViewDimension();
169     min.width = Math.max(min.width, view.width);
170     min.height = Math.max(min.height, view.height);
171
172     if (!min.equals(getPreferredSize())) {
173         setPreferredSize(min);
174         revalidate();
175     }
176 }
177
178 public TabbedPane getParentTabbedPane() {
179     return parentTabbedPane;
180 }
181
182 @Override
```

```
183     public String toString() {
184         return getTitle();
185     }
186
187     public ScrollPane getScrollPane() {
188         return scrollPane;
189     }
190
191     public static class InvalidTitleException extends Exception{
192         public InvalidTitleException(String s) {
193             super(s);
194         }
195     }
196
197     public void setTitle(String title) throws InvalidTitleException {
198         if (!UtilRegex.isIdentifier(title))
199             throw new InvalidTitleException("Invalid title "+title+"");
200         validateDuplicateTitle(title);
201         tabbedPanelTitle = title;
202         for (TitleChangedListener t : titleChangedListeners)
203             t.titleChanged(title);
204     }
205
206     public String getTitle() {
207         return tabbedPanelTitle;
208     }
209
210     public void addTitleChangedListener(TitleChangedListener t) {
211         titleChangedListeners.add(t);
212     }
213
214     public void removeTitleChangedListener(TitleChangedListener t) {
215         titleChangedListeners.remove(t);
216     }
217
218     private Polygon getSideArrow(int x, int y, double angle) {
219         AffineTransform trans = new AffineTransform();
220         trans.rotate(angle);
221         Point2D[] dest = new Point2D[3];
222         trans.transform(new Point[]{new Point(-5, -5), new Point(5,-5), new Point(0, 5)}, 0, dest,
223         0, 3);
224
225         Polygon arrow = new Polygon();
226         arrow.addPoint((int)Math.round(dest[0].getX()) + x, (int)Math.round(dest[0].getY()) + y);
227         arrow.addPoint((int)Math.round(dest[1].getX()) + x, (int)Math.round(dest[1].getY()) + y);
228         arrow.addPoint((int)Math.round(dest[2].getX()) + x, (int)Math.round(dest[2].getY()) + y);
229     }
230
231     private Ellipse2D[] getSideOvals() {
232         Ellipse2D[] ret = new Ellipse2D[8];
233         Rectangle view = scrollPane.getViewport().getViewRect();
234
235         ret[0] = new Ellipse2D.Double(view.x-halfCorner, view.y-halfCorner, cornerDia, cornerDia);
```

```
236         ret[1] = new Ellipse2D.Double(view.x+view.width/2-halfSide, view.y-halfSide, sideDia,
237             sideDia);
238         ret[2] = new Ellipse2D.Double(view.x+view.width-halfCorner, view.y-halfCorner, cornerDia,
239             cornerDia);
240         ret[3] = new Ellipse2D.Double(view.x-halfSide, view.y+view.height/2-halfSide, sideDia,
241             sideDia);
242         ret[4] = new Ellipse2D.Double(view.x+view.width-halfSide, view.y+view.height/2-halfSide,
243             sideDia, sideDia);
244         ret[5] = new Ellipse2D.Double(view.x-halfCorner, view.y+view.height-halfCorner, cornerDia,
245             cornerDia);
246         ret[6] = new Ellipse2D.Double(view.x+view.width/2-halfSide, view.y+view.height-halfSide,
247             sideDia, sideDia);
248         ret[7] = new Ellipse2D.Double(view.x+view.width-halfCorner, view.y+view.height-halfCorner,
249             cornerDia, cornerDia);
250     }
251
252     private int getSideOvalIndex(Point untransformedPoint) {
253         Rectangle view = scrollPane.getViewport().getViewRect();
254         if (!view.contains(untransformedPoint)) {
255             return -1;
256         }
257
258         Ellipse2D[] ovals = getSideOvals();
259         for (int i = 0; i < ovals.length; i++) {
260             if (ovals[i].contains(untransformedPoint))
261                 return i;
262         }
263
264         return -1;
265     }
266
267     protected boolean isOverSideOval(Point untransformedPoint) {
268         return getSideOvalIndex(untransformedPoint) != -1;
269     }
270
271     private synchronized boolean markSideOval(Point untransformedPoint) {
272         int index = getSideOvalIndex(untransformedPoint);
273         markedSideOvalIndex.set(index);
274         if (index != -1) {
275             markedSideOvalPoint = untransformedPoint;
276             return true;
277         }
278         markedSideOvalPoint = null;
279         return false;
280     }
281
282     private synchronized void unmarkSideOval() {
283         markedSideOvalIndex.set(-1);
284         markedSideOvalPoint = null;
285     }
286
287     private void moveLeft(Rectangle view, Dimension size) {
288         int move = Math.min(scrollStep, view.width);
289         if (view.x < move) {
```

```
284         move = view.x;
285     }
286     view.x -= move;
287     Dimension minDim = transformDimension(getTransform(), getMinimumDimension());
288     if (size.width > minDim.width) {
289         if (minDim.width > view.x+view.width)
290             size.width = minDim.width;
291         else
292             size.width = view.x+view.width;
293     }
294 }
295
296 private void moveUp(Rectangle view, Dimension size) {
297     int move = Math.min(scrollStep, view.height);
298     if (view.y < move) {
299         move = view.y;
300     }
301     view.y -= move;
302     Dimension minDim = transformDimension(getTransform(), getMinimumDimension());
303     if (size.height > minDim.height) {
304         if (minDim.height > view.y+view.height)
305             size.height = minDim.height;
306         else
307             size.height = view.y+view.height;
308     }
309 }
310
311 private void moveRight(Rectangle view, Dimension size) {
312     int x = view.x + view.width + scrollStep + 30;
313     if (x > size.width) {
314         size.width = x;
315     }
316     view.x += scrollStep;
317 }
318
319 private void moveDown(Rectangle view, Dimension size) {
320     int y = view.y + view.height + scrollStep + 30;
321     if (y > size.height) {
322         size.height = y;
323     }
324     view.y += scrollStep;
325 }
326
327 private Dimension getViewDimension() {
328     Rectangle view = scrollPane.getViewport().getViewRect();
329     return new Dimension(view.x+view.width, view.y+view.height);
330 }
331
332 private synchronized void moveScreen() {
333     Dimension pref = getPreferredSize();
334     Rectangle view = scrollPane.getViewport().getViewRect();
335     pref.width = Math.max(pref.width, view.width);
336     pref.height = Math.max(pref.height, view.height);
337
338     switch (markedSideOvalIndex.get()) {
```

```
339     case -1:
340         return;
341     case 0:
342         moveUp(view, pref);
343         moveLeft(view, pref);
344         break;
345     case 1:
346         moveUp(view, pref);
347         break;
348     case 2:
349         moveUp(view, pref);
350         moveRight(view, pref);
351         break;
352     case 3:
353         moveLeft(view, pref);
354         break;
355     case 4:
356         moveRight(view, pref);
357         break;
358     case 5:
359         moveDown(view, pref);
360         moveLeft(view, pref);
361         break;
362     case 6:
363         moveDown(view, pref);
364         break;
365     case 7:
366         moveDown(view, pref);
367         moveRight(view, pref);
368         break;
369     default:
370         throw new RuntimeException("invalid direction");
371     }
372
373     Rectangle firstView = scrollPane.getViewport().getViewRect();
374     setPreferredSize(pref);
375     revalidate();
376     this.scrollRectToVisible(view);
377     if (markedSideOvalPoint != null) {
378         Rectangle secondView = scrollPane.getViewport().getViewRect();
379         int x = secondView.x - firstView.x;
380         int y = secondView.y - firstView.y;
381         markedSideOvalPoint.x += x;
382         markedSideOvalPoint.y += y;
383         applyMouseDrag(markedSideOvalPoint);
384     }
385 }
386
387 public void drawBorder(Graphics g) {
388     Graphics2D g2 = (Graphics2D)g;
389     g2.transform(getInverseTransform());
390
391     Rectangle view = scrollPane.getViewport().getViewRect();
392
393     g2.setColor(ColorTheme.HOVERED);
```

```
394     for (Ellipse2D e : getSideOvals())
395         g2.fill(e);
396
397     int index = markedSideOvalIndex.get();
398     if (index == 0)
399         g2.setColor(ColorTheme.SELECTED);
400     else
401         g2.setColor(ColorTheme.HOVERED);
402     g2.fill(getSideArrow(view.x+quardCorner, view.y+quardCorner, Math.PI - Math.PI/4));
403     if (index == 1)
404         g2.setColor(ColorTheme.SELECTED);
405     else
406         g2.setColor(ColorTheme.HOVERED);
407     g2.fill(getSideArrow(view.x+view.width/2, view.y+quardSide, Math.PI));
408     if (index == 2)
409         g2.setColor(ColorTheme.SELECTED);
410     else
411         g2.setColor(ColorTheme.HOVERED);
412     g2.fill(getSideArrow(view.x+view.width-quardCorner, view.y+quardCorner, Math.PI + Math.PI/4));
413     if (index == 3)
414         g2.setColor(ColorTheme.SELECTED);
415     else
416         g2.setColor(ColorTheme.HOVERED);
417     g2.fill(getSideArrow(view.x+quardSide, view.y+view.height/2, Math.PI/2));
418     if (index == 4)
419         g2.setColor(ColorTheme.SELECTED);
420     else
421         g2.setColor(ColorTheme.HOVERED);
422     g2.fill(getSideArrow(view.x+view.width-quardSide, view.y+view.height/2, Math.PI + Math.PI/2));
423     if (index == 5)
424         g2.setColor(ColorTheme.SELECTED);
425     else
426         g2.setColor(ColorTheme.HOVERED);
427     g2.fill(getSideArrow(view.x+quardCorner, view.y+view.height-quardCorner, Math.PI/4));
428     if (index == 6)
429         g2.setColor(ColorTheme.SELECTED);
430     else
431         g2.setColor(ColorTheme.HOVERED);
432     g2.fill(getSideArrow(view.x+view.width/2, view.y+view.height-quardSide, 0));
433     if (index == 7)
434         g2.setColor(ColorTheme.SELECTED);
435     else
436         g2.setColor(ColorTheme.HOVERED);
437     g2.fill(getSideArrow(view.x+view.width-quardCorner, view.y+view.height-quardCorner, Math.PI + Math.PI/4 + Math.PI/2));
438 }
439
440 public abstract Icon getIcon();
441
442 public class ScrollPane extends UtilScrollPane {
443     protected ScrollPane() {
444         super(TabbedPane.this);
445         setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
```

```
446         setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
447     }
448     public TabbedPane getTabbedPane() {
449         return TabbedPane.this;
450     }
451 }
452
453     public static interface TitleChangedListener {
454         void titleChanged(String title);
455     }
456
457     public AffineTransform getTransform() {
458         return zoomTransform;
459     }
460
461     public AffineTransform getInverseTransform() {
462         try {
463             return zoomTransform.createInverse();
464         } catch (NoninvertibleTransformException e) {
465             throw new RuntimeException("invalid zoom");
466         }
467     }
468
469     public int inverseZoom(double x) {
470         return applyZoom(getInverseTransform(), x);
471     }
472
473     public int applyZoom(AffineTransform trans, double x) {
474         return (int) Math.round(x*trans.getScaleX());
475     }
476
477     public Dimension transformDimension(AffineTransform trans, Dimension d) {
478         int width = applyZoom(trans, d.width);
479         int height = applyZoom(trans, d.height);
480         return new Dimension(width, height);
481     }
482
483     public Rectangle transformRectangle(AffineTransform trans, Rectangle r) {
484         Point loc = transformPoint(trans, r.getLocation());
485         int width = applyZoom(trans, r.width);
486         int height = applyZoom(trans, r.height);
487         return new Rectangle(loc.x, loc.y, width, height);
488     }
489
490     public Point transformPoint(AffineTransform trans, Point p) {
491         Point2D p2 = trans.transform(p, null);
492         return new Point((int) Math.round(p2.getX()), (int) Math.round(p2.getY()));
493     }
494
495     public Point inverseTransformPoint(Point p) {
496         return transformPoint(getInverseTransform(), p);
497     }
498
499     public MouseEvent inverseTransformMouseEvent(MouseEvent e) {
500         AffineTransform trans = getInverseTransform();
```

```
501     Point p = transformPoint(trans, e.getPoint());
502     MouseEvent ret = new MouseEvent(e.getComponent(), e.getID(), e.getWhen(),
503         e.getModifiers(), p.x, p.y, e.getClickCount(),
504         e.isPopupTrigger(), e.getButton());
505     return ret;
506 }
507
508 public void zoomIn() {
509     //Rectangle rec = scrollPane.setViewport().getViewRect();
510     //zoomTransform.translate(rec.width*zoomInScale/2, rec.height*zoomInScale/2);
511     zoomTransform.scale(zoomInScale, zoomInScale);
512     //zoomTransform.translate(-rec.width*zoomInScale/2, -rec.height*zoomInScale/2);
513     repaint();
514 }
515
516 public void zoomOut() {
517     //Rectangle rec = scrollPane.setViewport().getViewRect();
518     //zoomTransform.translate(rec.width*zoomInScale/2, rec.height*zoomInScale/2);
519     zoomTransform.scale(zoomOutScale, zoomOutScale);
520     //zoomTransform.translate(-rec.width*zoomInScale/2, -rec.height*zoomInScale/2);
521     repaint();
522 }
523
524 public abstract Predicate<DataPartParameter> getChildParamaterDeleteEvent();
525 public abstract Predicate<DataPartResult> getChildResultDeleteEvent();
526 public abstract Dimension getMinimumDimension();
527 public abstract void applyMouseDrag(Point p1);
528 public abstract void validateDuplicateTitle(String title) throws InvalidTitleException;
529 public abstract boolean hasDataFlowWithTitle(String title);
530 public abstract boolean isParameterConnected(DataFlowPanel containingPanel,
531     DataPartParameter dataPartParameter);
532 public abstract void parameterTypeChanged(DataFlowPanel containingPanel,
533     DataPartParameter dataPartParameter);
534 public abstract void changeParameterName(DataFlowPanel containingPanel,
535     DataPartParameter dataPartParameter, String oldName);
536 public abstract void changeResultName(DataFlowPanel containingPanel,
537     DataPartResult dataPartResult, String oldName);
538 public abstract boolean isResultConnected(DataFlowPanel containingPanel,
539     DataPartResult dataPartResult);
540 public abstract void resultTypeChanged(DataFlowPanel containingPanel,
541     DataPartResult dataPartResult);
542 public abstract void childFailStateChanged();
543
544 public void scrollToPoint(Point p) {
545     Rectangle view = scrollPane.setViewport().getViewRect();
546     int halfHeight = view.height/2;
547     int halfWidth = view.width/2;
548     Rectangle rect = new Rectangle(p.x-halfWidth, p.y-halfHeight, view.width, view.height);
549     scrollRectToVisible(rect);
550 }
551
552 public abstract void scrollToComponent(int seqNumber);
553 public abstract void scrollToPath(int startSeq, int eventIndex, int endSeq);
554
555 Rectangle getDrawRectangle(Rectangle dragRectangle) {
```

```
556     int x = dragRectangle.x;
557     int y = dragRectangle.y;
558     int w = dragRectangle.width;
559     int h = dragRectangle.height;
560     if (w < 0) {
561         w = Math.abs(w);
562         x -= w;
563     }
564     if (h < 0) {
565         h = Math.abs(h);
566         y -= h;
567     }
568     return new Rectangle(x, y, w, h);
569 }
570 }
```

LISTING E.100: gui/TabbedPane.java

```
1 package gui;
2
3 import java.awt.BasicStroke;
4 import java.awt.BorderLayout;
5 import java.awt.Color;
6 import java.awt.Component;
7 import java.awt.FlowLayout;
8 import java.awt.Graphics;
9 import java.awt.Graphics2D;
10 import java.awt.GridBagConstraints;
11 import java.awt.GridBagLayout;
12 import java.awt.GridLayout;
13 import java.awt.MouseInfo;
14 import java.awt.Point;
15 import java.awt.Rectangle;
16 import java.awt.Stroke;
17 import java.awt.event.ActionEvent;
18 import java.awt.event.ActionListener;
19 import java.awt.event.MouseAdapter;
20 import java.awt.event.MouseEvent;
21 import java.awt.event.MouseListener;
22 import java.awt.event.MouseMotionListener;
23 import java.awt.event.MouseWheelEvent;
24 import java.awt.event.MouseWheelListener;
25 import java.awt.geom.Line2D;
26 import java.awt.geom.Point2D;
27 import java.util.ArrayList;
28 import java.util.List;
29
30 import javax.swing.BorderFactory;
31 import javax.swing.JComponent;
32 import javax.swing.JDialog;
33 import javax.swing.JMenuItem;
34 import javax.swing.JOptionPane;
35 import javax.swing.JPanel;
36 import javax.swing.JPopupMenu;
37 import javax.swing.JScrollBar;
```

```
38 import javax.swing.JScrollPane;
39 import javax.swing.JTextArea;
40 import javax.swing.SwingUtilities;
41 import javax.swing.border.TitledBorder;
42 import javax.swing.text.DefaultCaret;
43
44 import lang.type.RecordColumn;
45 import lang.type.TypeAuto;
46 import lang.type.TypeBool;
47 import lang.type.TypeConnection;
48 import lang.type.TypeError;
49 import lang.type.TypeList;
50 import lang.type.TypeNumber;
51 import lang.type.TypeRecord;
52 import lang.type.TypeString;
53 import lang.type.TypeTime;
54 import lang.type.TypeUnknown;
55 import util.ColorTheme;
56 import util.UtilFont;
57
58 @SuppressWarnings("serial")
59 public class TypeConnectionPanel extends JPanel {
60     private GlassPanel glassPanel;
61     private TypeConnection typeConnection;
62     private JDialog window;
63     private Container container;
64
65     public boolean isConnected() {
66         return typeConnection.isConnected();
67     }
68
69     private class GlassLine {
70         private PrimitiveTextArea outArea;
71         private PrimitiveTextArea inArea;
72         public GlassLine(PrimitiveTextArea outArea, PrimitiveTextArea inArea) {
73             this.outArea = outArea;
74             this.inArea= inArea;
75         }
76         public Line2D getLine() {
77             return new Line2D.Double(outArea.getCenter(), inArea.getCenter());
78         }
79     }
80
81     static class Container extends JPanel {
82         private TypeConnectionPanel connPanel;
83
84         Container(TypeConnectionPanel p) {
85             setLayout(new BorderLayout());
86             connPanel = p;
87             add(p);
88             p.container = this;
89             p.glassPanel.container = this;
90         }
91
92         private void reset(TypeConnectionPanel p) {
```

```
93         remove(connPanel);
94         add(p);
95         connPanel = p;
96         p.container = this;
97         p.revalidate();
98         p.repaint();
99     }
100
101    public TypeConnectionPanel get() {
102        return connPanel;
103    }
104}
105
106 static class GlassPanel extends JComponent implements MouseMotionListener, MouseListener,
107     MouseWheelListener {
108     private ArrayList<GlassLine> lines = new ArrayList<>();
109     private Line2D dragLine;
110     private Rectangle hoverRectangle;
111     private Component prevComponent;
112     private int highlightLine = -1;
113     private int clickLine = -1;
114     private int deleteLine = -1;
115     private JPopupMenu popupMenu;
116     private JDialog window;
117     private JScrollPane scrollPane;
118     private JScrollBar pressedScrollBar;
119     private Container container;
120
121     public GlassPanel(JDialog window, JScrollPane scrollPane) {
122         this.window = window;
123         this.scrollPane = scrollPane;
124
125         addMouseListener(this);
126         addMouseMotionListener(this);
127         addMouseWheelListener(this);
128
129         popupMenu = new JPopupMenu();
130         JMenuItem buttonDelete = new JMenuItem("Delete");
131         popupMenu.add(buttonDelete);
132         buttonDelete.addActionListener(new ActionListener() {
133             @Override
134             public void actionPerformed(ActionEvent e) {
135                 deleteEvent();
136             }
137         });
138     }
139
140     private void reset() {
141         lines = new ArrayList<>();
142         dragLine = null;
143         highlightLine = -1;
144         clickLine = -1;
145         deleteLine = -1;
146         hoverRectangle = null;
147     }
148 }
```

```
147
148     private void maybeShowPopup(MouseEvent e) {
149         int line = getLineIndex(e.getPoint());
150         if (e.isPopupTrigger() && line >= 0) {
151             deleteLine = line;
152             popupMenu.show(this, e.getX(), e.getY());
153         }
154     }
155
156     private void deleteEvent() {
157         if (deleteLine != -1) {
158             boolean reval = false;
159
160             GlassLine line = lines.get(deleteLine);
161             int outIndex = line.outArea.index;
162             int inIndex = line.inArea.index;
163             if (line.outArea.typeConnection.removeConnection(
164                 new TypeConnection.SingleConnection(outIndex, inIndex))) {
165                 reval = true;
166             }
167             lines.remove(deleteLine);
168             clickLine = -1;
169             highlightLine = -1;
170             deleteLine = -1;
171
172             if (reval) {
173                 container.get().revalidateTypeConnection();
174             } else {
175                 this.repaint();
176             }
177         }
178     }
179
180     @Override
181     public void paintComponent(Graphics g) {
182         super.paintComponent(g);
183         Graphics2D g2 = (Graphics2D)g;
184         Stroke orig = g2.getStroke();
185         if (hoverRectangle != null) {
186             g2.setStroke(new BasicStroke(3));
187             g2.setColor(ColorTheme.HOVERED);
188             g2.draw(hoverRectangle);
189
190         }
191         if (highlightLine >= 0) {
192             g2.setStroke(new BasicStroke(6));
193             g2.setColor(ColorTheme.HOVERED);
194             g2.draw(lines.get(highlightLine).getLine());
195         }
196         if (clickLine >= 0) {
197             g2.setStroke(new BasicStroke(4));
198             g2.setColor(ColorTheme.SELECTED);
199             g2.draw(lines.get(clickLine).getLine());
200         }
201         g2.setStroke(new BasicStroke(2));
```

```
202         g2.setColor(new Color(0, 0, 0, 127));
203         if (dragLine != null) {
204             g2.draw(dragLine);
205         }
206         for (GlassLine line : lines)
207             g2.draw(line.getLine());
208         g2.setStroke(orig);
209     }
210
211     public void setHoverComponent(ConnectTextArea comp) {
212         Point compPoint = SwingUtilities.convertPoint(this, new Point(0,0), comp.
213             getConvertComponent());
214         compPoint.x = -compPoint.x;
215         compPoint.y = -compPoint.y;
216         Rectangle d = comp.getAreaBounds();
217         hoverRectangle = new Rectangle(compPoint.x, compPoint.y, d.width, d.height);
218     }
219
220     public void unsetHoverComponent() {
221         hoverRectangle = null;
222     }
223
224     private int getLineIndex(Point point) {
225         for (int idx = 0; idx < lines.size(); idx++) {
226             Line2D line = lines.get(idx).getLine();
227             if (line.intersects(point.x-3, point.y-3, 6, 6)) {
228                 return idx;
229             }
230         }
231         return -1;
232     }
233
234     private void setHighlightLine(MouseEvent e) {
235         highlightLine = getLineIndex(e.getPoint());
236     }
237
238     private void setClickLine(MouseEvent e) {
239         clickLine = getLineIndex(e.getPoint());
240     }
241
242     @Override
243     public void mouseDragged(MouseEvent e) {
244         Component c = resendMouseMoved(e);
245         if (!(c instanceof PrimitiveTextArea)) {
246             setHighlightLine(e);
247         } else {
248             highlightLine = -1;
249         }
250         if (dragLine != null) {
251             dragLine.setLine(dragLine.getPt1(), e.getPoint());
252         }
253         this.repaint();
254     }
255
256     @Override
```

```
256     public void mouseMoved(MouseEvent e) {
257         Component c = resendMouseMoved(e);
258         if (!(c instanceof PrimitiveTextArea)) {
259             setHighlightLine(e);
260         } else {
261             highlightLine = -1;
262         }
263         this.repaint();
264     }
265
266     @Override
267     public void mouseClicked(MouseEvent e) {
268     }
269
270     @Override
271     public void mousePressed(MouseEvent e) {
272         Component c = resendMouseEvent(e, MouseEvent.MOUSE_PRESSED);
273         if (!(c instanceof PrimitiveTextArea)) {
274             setClickLine(e);
275         } else {
276             clickLine = -1;
277         }
278         this.repaint();
279         maybeShowPopup(e);
280     }
281
282     @Override
283     public void mouseReleased(MouseEvent e) {
284         resendMouseEvent(e, MouseEvent.MOUSE_RELEASED);
285         dragLine = null;
286         this.repaint();
287         maybeShowPopup(e);
288     }
289
290     @Override
291     public void mouseEntered(MouseEvent e) {}
292
293     @Override
294     public void mouseExited(MouseEvent e) {}
295
296     private Component resendMouseMoved(MouseEvent e) {
297         Point point = e.getPoint();
298         Component component = SwingUtilities.getDeepestComponentAt(
299                 window.getContentPane(),
300                         point.x,
301                         point.y);
302         if (component != null && component != this && prevComponent != component &&
303             !(component instanceof JScrollPane))) {
304             Point componentPoint = SwingUtilities.convertPoint(this, point, component);
305             MouseEvent exitEvent = null;
306             if (prevComponent != null)
307                 exitEvent = new MouseEvent(prevComponent, MouseEvent.MOUSE_EXITED,
308                                         e.getWhen(), e.getModifiers(), componentPoint.x, componentPoint.y,
309                                         e.getClickCount(), e.isPopupTrigger());
310             MouseEvent enterEvent = new MouseEvent(component, MouseEvent.MOUSE_ENTERED,
```

```
311             e.getWhen(), e.getModifiers(), componentPoint.x, componentPoint.y,
312             e.getClickCount(), e.isPopupTrigger());
313         if (prevComponent != null)
314             prevComponent.dispatchEvent(exitEvent);
315         component.dispatchEvent(enterEvent);
316         prevComponent = component;
317     }
318
319     if (pressedScrollBar != null) {
320         Point componentPoint = SwingUtilities.convertPoint(this, point, pressedScrollBar);
321         MouseEvent mouseEvent = new MouseEvent(pressedScrollBar, e.getID(), e.getWhen(),
322             e.getModifiers(), componentPoint.x, componentPoint.y, e.getClickCount(), e
323             .isPopupTrigger());
324         pressedScrollBar.dispatchEvent(mouseEvent);
325     }
326
327     return component;
328 }
329
330 private Component resendMouseEvent(MouseEvent e, int id) {
331     Point point = e.getPoint();
332     Component component = SwingUtilities.getDeepestComponentAt(
333         window.getContentPane(),
334         point.x,
335         point.y);
336     if (component != null && component != this) {
337         Point componentPoint = SwingUtilities.convertPoint(this, point, component);
338         MouseEvent event = new MouseEvent(component, id, e.getWhen(),
339             e.getModifiers(), componentPoint.x, componentPoint.y, e.getClickCount(), e
340             .isPopupTrigger());
341         component.dispatchEvent(event);
342     }
343     pressedScrollBar = null;
344     if (scrollPane != null && id == MouseEvent.MOUSE_PRESSED) {
345         Point p = SwingUtilities.convertPoint(this, point, scrollPane.getVerticalScrollBar()
346         ());
347         if (p.x > 0) {
348             pressedScrollBar = scrollPane.getVerticalScrollBar();
349         } else {
350             p = SwingUtilities.convertPoint(this, point, scrollPane.getVerticalScrollBar()
351         );
352             if (p.y > 0) {
353                 pressedScrollBar = scrollPane.getHorizontalScrollBar();
354             }
355         }
356     }
357     return component;
358 }
359
360 private ConnectTextArea getDragLineTextArea() {
361     Point2D p1 = dragLine.getPt1();
362     Component component = SwingUtilities.getDeepestComponentAt(
363         window.getContentPane(),
364         (int)p1.getX(),
365         (int)p1.getY());
```

```
362         return (ConnectTextArea)component;
363     }
364
365     @Override
366     public void mouseWheelMoved(MouseWheelEvent e) {
367         if (scrollPane == null)
368             return;
369         JScrollBar bar = scrollPane.getVerticalScrollBar();
370         Point componentPoint = SwingUtilities.convertPoint(this, e.getPoint(), bar);
371         MouseWheelEvent next = new MouseWheelEvent(bar, e.getID(), e.getWhen(), e.
372             getModifiersEx(),
373             componentPoint.x, componentPoint.y, e.getClickCount(), e.isPopupTrigger(),
374             e.getScrollType(), e.getScrollAmount(), e.getWheelRotation());
375         bar.dispatchEvent(next);
376     }
377
378     private class HoverListener extends MouseAdapter {
379         private ConnectTextArea comp;
380         public HoverListener(ConnectTextArea comp) {
381             this.comp = comp;
382         }
383
384         @Override
385         public void mouseEntered(MouseEvent e) {
386             glassPanel.setHoverComponent(comp);
387         }
388
389         @Override
390         public void mouseExited(MouseEvent e) {
391             glassPanel.unsetHoverComponent();
392         }
393
394         @Override
395         public void mousePressed(MouseEvent e) {
396             Point compCenter = comp.getCenter();
397             glassPanel.dragLine = new Line2D.Double(compCenter, compCenter);
398         }
399
400         private void showErrorMessage(String msg, String title) {
401             glassPanel.unsetHoverComponent();
402             JOptionPane.showMessageDialog(window, msg,
403                 title, JOptionPane.ERROR_MESSAGE);
404
405             Point p = MouseInfo.getPointerInfo().getLocation();
406             Point loc = glassPanel.getLocationOnScreen();
407             p.x -= loc.x;
408             p.y -= loc.y;
409             p = SwingUtilities.convertPoint(glassPanel, p, comp);
410             if (comp.getAreaBounds().contains(p)) {
411                 glassPanel.setHoverComponent(comp);
412             }
413         }
414
415         @Override
```

```
416     public void mouseReleased(MouseEvent e) {
417         if (glassPanel.dragLine == null)
418             return;
419
420         ConnectTextArea start1 = glassPanel.getDragLineTextArea();
421
422         if (start1.isOutput == comp.isOutput) {
423             glassPanel.dragLine = null;
424             return;
425         }
426
427         if (start1 instanceof PrimitiveTextArea && comp instanceof PrimitiveTextArea) {
428             singleConnect((PrimitiveTextArea)start1);
429         } else if (!(start1 instanceof AutoTextArea) && start1.getClass() == comp.getClass())
430         ||
431             !(start1 instanceof PrimitiveTextArea) && start1.isOutput && comp instanceof
432             AutoTextArea ||
433                 !(comp instanceof PrimitiveTextArea) && comp.isOutput && start1 instanceof
434             AutoTextArea) {
435                     multiConnect(start1);
436             } else if (start1 instanceof PrimitiveTextArea && start1.isOutput && comp instanceof
437             AutoTextArea ||
438                 comp instanceof PrimitiveTextArea && comp.isOutput && start1 instanceof
439             AutoTextArea) {
440                     singleToMultiConnect(start1);
441             } else if (start1 instanceof PrimitiveTextArea && !start1.isOutput && comp instanceof
442             AutoTextArea ||
443                 comp instanceof PrimitiveTextArea && !comp.isOutput && start1 instanceof
444             AutoTextArea) {
445                     multiToSingleConnect(start1);
446             } else if (comp instanceof AutoTextArea || start1 instanceof AutoTextArea) {
447                     multiConnect(start1);
448             } else {
449                     showErrorMessage("Invalid connection", "Connection fail");
450             }
451         }
452
453         private void multiConnect(ConnectTextArea start) {
454             glassPanel.dragLine = null;
455             try {
456                 int outIndex = start.isOutput ? start.index : comp.index;
457                 int inIndex = start.isOutput ? comp.index : start.index;
458                 typeConnection.addMultiConnection(new TypeConnection.MultiConnection(outIndex,
459                     inIndex));
460             } catch (IllegalArgumentException ex) {
461                 showErrorMessage(ex.getMessage(), "Connection fail");
462                 return;
463             }
464             revalidateTypeConnection();
465         }
466
467         private void multiToSingleConnect(ConnectTextArea start) {
468             glassPanel.dragLine = null;
469             try {
470                 int outIndex = start.isOutput ? start.index : comp.index;
```

```

463         int inIndex = start.isOutput ? comp.index : start.index;
464         typeConnection.addMultiToSingleConnection(new TypeConnection.
465             MultiToSingleConnection(outIndex, inIndex));
466     } catch (IllegalArgumentException ex) {
467         showErrorMessage(ex.getMessage(), "Connection fail");
468         return;
469     }
470     revalidateTypeConnection();
471 }
472
473 private void singleToMultiConnect(ConnectTextArea start) {
474     glassPanel.dragLine = null;
475     try {
476         int outIndex = start.isOutput ? start.index : comp.index;
477         int inIndex = start.isOutput ? comp.index : start.index;
478         typeConnection.addSingleToMultiConnection(new TypeConnection.
479             SingleToMultiConnection(outIndex, inIndex));
480     } catch (IllegalArgumentException ex) {
481         showErrorMessage(ex.getMessage(), "Connection fail");
482         return;
483     }
484     revalidateTypeConnection();
485 }
486
487 private void singleConnect(PrimitiveTextArea start) {
488     try {
489         int outIndex = start.isOutput ? start.index : comp.index;
490         int inIndex = start.isOutput ? comp.index : start.index;
491         typeConnection.addConnection(new TypeConnection.SingleConnection(outIndex, inIndex
492             ));
493     } catch (IllegalArgumentException ex) {
494         showErrorMessage(ex.getMessage(), "Connection fail");
495         glassPanel.dragLine = null;
496         return;
497     }
498
499     Point compCenter = comp.getCenter();
500     glassPanel.dragLine = new Line2D.Double(glassPanel.dragLine.getP1(), compCenter);
501     GlassLine line;
502     if (start.isOutput)
503         line = new GlassLine(start, (PrimitiveTextArea)comp);
504     else
505         line = new GlassLine((PrimitiveTextArea)comp, start);
506     glassPanel.lines.add(line);
507     glassPanel.dragLine = null;
508 }
509
510 public static TypeConnectionPanel.Container get(TypeConnection conn, JDialog window,
511     GlassPanel glassPanel) {
512     TypeConnectionPanel p = new TypeConnectionPanel(conn, window, glassPanel);
513     Container c = new Container(p);
514     return c;
515 }
```

```

515     private TypeConnectionPanel(TypeConnection conn, JDialog window, GlassPanel glassPanel) {
516         this.window = window;
517         this.glassPanel = glassPanel;
518         this.typeConnection = conn;
519         initialize();
520     }
521
522     private void initialize() {
523         GridLayout layout = new GridLayout(2,1);
524         setLayout(layout);
525
526         JPanel out = new JPanel(new FlowLayout(FlowLayout.CENTER, 20, 20));
527         out.setBackground(Color.white);
528         JPanel outType = new JPanel(new BorderLayout());
529         ArrayList<PrimitiveTextArea> outConns = new ArrayList<>();
530         outType.add(getTypeGraphics(typeConnection.getOutputType(), true, new IntReference(),
531             new IntReference(), outConns, typeConnection));
532         outType.setBorder(BorderFactory.createTitledBorder(BorderFactory.createEmptyBorder(),
533             "Output", TitledBorder.CENTER, TitledBorder.TOP));
534         out.add(outType);
535         add(out);
536
537         JPanel in = new JPanel(new FlowLayout(FlowLayout.CENTER, 20, 20));
538         in.setBackground(Color.white);
539         JPanel inType = new JPanel(new BorderLayout());
540         ArrayList<PrimitiveTextArea> inConns = new ArrayList<>();
541         inType.add(getTypeGraphics(typeConnection.getInputType(), false, new IntReference(),
542             new IntReference(), inConns, typeConnection));
543         inType.setBorder(BorderFactory.createTitledBorder(BorderFactory.createEmptyBorder(),
544             "Input", TitledBorder.CENTER, TitledBorder.BOTTOM));
545         in.add(inType);
546         add(in);
547
548         for (PrimitiveTextArea outArea : outConns) {
549             for (PrimitiveTextArea inArea : inConns) {
550                 if (typeConnection.hasConnection(outArea.index, inArea.index))
551                     glassPanel.lines.add(new GlassLine(outArea, inArea));
552             }
553         }
554     }
555
556     private void revalidateTypeConnection() {
557         glassPanel.reset();
558         this.container.reset(new TypeConnectionPanel(typeConnection, window, glassPanel));
559     }
560
561     private static class IntReference {
562         int value = 0;
563     }
564
565     private JPanel getTypeGraphics(lang.type.Type type, boolean isOutput, IntReference ref,
566         IntReference sref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
567     {
568         switch (type.getTypeCode()) {
569             case BOOL:

```

```

569         return getBoolGraphics((TypeBool)type, isOutput, ref, sref, connectAreas, typeConn);
570     case AUTO:
571         return getAutoGraphics((TypeAuto)type, isOutput, ref, sref, connectAreas, typeConn);
572     case LIST:
573         return getListGraphics((TypeList)type, isOutput, ref, sref, connectAreas, typeConn);
574     case NUMBER:
575         return getNumberGraphics((TypeNumber)type, isOutput, ref, sref, connectAreas, typeConn
576     );
576     case RECORD:
577         return getRecordGraphics((TypeRecord)type, isOutput, ref, sref, connectAreas, typeConn
578     );
578     case STRING:
579         return getStringGraphics((TypeString)type, isOutput, ref, sref, connectAreas, typeConn
580     );
580     case TIME:
581         return getTimeGraphics((TypeTime)type, isOutput, ref, sref, connectAreas, typeConn);
582     case UNKNOWN:
583         return getUnknownGraphics((TypeUnknown)type, isOutput, ref, sref, connectAreas,
584     typeConn);
584     case ERROR:
585         return getErrorGraphics((TypeError)type, isOutput, ref, sref, connectAreas, typeConn);
586     default:
587         throw new RuntimeException("unexpected type code");
588     }
589 }
590
591 private abstract class ConnectTextArea extends JTextArea {
592     boolean isOutput;
593     int index;
594     TypeConnection typeConnection;
595     ConnectTextArea(int w, int h, boolean isOutput, IntReference ref, TypeConnection typeConn)
596     {
597         super(w,h);
598         this.typeConnection = typeConn;
599         this.isOutput = isOutput;
600         this.index = ref.value;
601
602         this.setFont(UtilFont.textFont);
603         this.setEditable(false);
604         DefaultCaret caret = (DefaultCaret)this.getCaret();
605         caret.setUpdatePolicy(DefaultCaret.ALWAYS_UPDATE);
606         this.setBorder(BorderFactory.createLoweredBevelBorder());
607         this.addMouseListener(new HoverListener(this));
608     }
609
610     Point getCenter() {
611         Rectangle r = getAreaBounds();
612         Point compCenter = SwingUtilities.convertPoint(glassPanel, new Point(0,0),
613         getConvertComponent());
614         compCenter.x = -compCenter.x + r.width/2;
615         if (isOutput)
616             compCenter.y = -compCenter.y + r.height - 4;
617         else
618             compCenter.y = -compCenter.y + 4;
619         return compCenter;
620     }

```

```
618     }
619
620     Component getConvertComponent() {
621         return this;
622     }
623
624     abstract Rectangle getAreaBounds();
625 }
626
627     private class PrimitiveTextArea extends ConnectTextArea {
628         PrimitiveTextArea(int w, int h, boolean isOutput, IntReference ref, TypeConnection typeConn) {
629             super(w,h,isOutput,ref,typeConn);
630         }
631
632         Rectangle getAreaBounds() {
633             return getBounds();
634         }
635     }
636
637     private class RecordTextArea extends ConnectTextArea {
638         RecordTextArea prev;
639         RecordTextArea next;
640
641         RecordTextArea(int w, int h, boolean isOutput, IntReference ref, TypeConnection typeConn)
642         {
643             super(w, h, isOutput, ref, typeConn);
644         }
645
646         @Override
647         Rectangle getAreaBounds() {
648             RecordTextArea start = (RecordTextArea)getConvertComponent();
649             int w = 0;
650             RecordTextArea area = start;
651             do {
652                 w += area.getBounds().width;
653                 area = area.next;
654             } while (area != null);
655
656             Rectangle startBounds = start.getBounds();
657             int h = startBounds.height;
658             int x = startBounds.x;
659             int y = startBounds.y;
660
661             return new Rectangle(x, y, w, h);
662         }
663
664         void addColumnAfter(RecordTextArea col) {
665             col.prev = this;
666             this.next = col;
667         }
668
669         @Override
670         Component getConvertComponent() {
671             RecordTextArea start = this;
```

```
671         while (start.prev != null)
672             start = start.prev;
673         return start;
674     }
675 }
676
677     private class ListTextArea extends ConnectTextArea {
678         ListTextArea(int w, int h, boolean isOutput, IntReference ref, TypeConnection typeConn) {
679             super(w, h, isOutput, ref, typeConn);
680         }
681
682         @Override
683         Rectangle getAreaBounds() {
684             return getBounds();
685         }
686     }
687
688     private class AutoTextArea extends ConnectTextArea {
689         AutoTextArea(int w, int h, boolean isOutput, IntReference ref, TypeConnection typeConn) {
690             super(w, h, isOutput, ref, typeConn);
691         }
692
693         @Override
694         Rectangle getAreaBounds() {
695             return getBounds();
696         }
697     }
698
699     private ConnectTextArea getTextArea(int x, int y, String text, lang.type.Type type,
700                                         boolean isOutput, IntReference ref, TypeConnection typeConnection) {
701         ConnectTextArea area;
702         switch (type.getTypeCode()) {
703             case AUTO:
704                 area = new AutoTextArea(x,y,isOutput,ref,typeConnection);
705                 break;
706             case LIST:
707                 area = new ListTextArea(x,y,isOutput,ref,typeConnection);
708                 break;
709             case RECORD:
710                 area = new RecordTextArea(x,y,isOutput,ref,typeConnection);
711                 break;
712             case TIME:
713             case STRING:
714             case NUMBER:
715             case BOOL:
716             case UNKNOWN:
717             case ERROR:
718                 area = new PrimitiveTextArea(x,y,isOutput,ref,typeConnection);
719                 break;
720             default:
721                 throw new RuntimeException("unexpected type code?");
722         }
723
724         area.setText(text);
725         return area;
```

```
726     }
727
728     private JPanel getRecordGraphics(TypeRecord type, boolean isOutput, IntReference ref,
729             IntReference sref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
730     {
731         JPanel ret = new JPanel();
732
733         GridBagLayout columnLayout = new GridBagLayout();
734         GridBagConstraints constraints = new GridBagConstraints();
735         ret.setLayout(columnLayout);
736
737         List<RecordColumn> cols = type.getColumnsInOrder();
738         int xIdx = 0;
739         RecordTextArea prev = null;
740         int oldValue = sref.value;
741         sref.value += 1;
742         for (RecordColumn col : cols) {
743             RecordTextArea text = (RecordTextArea)getTextArea(1,6,col.getId(),type,isOutput,sref,
744             typeConn);
745             text.setBackground(new Color(0xe0, 0xff, 0xff));
746             text.index = oldValue;
747             JPanel child = getTypeGraphics(col.getType(),isOutput,ref,sref,connectAreas,typeConn);
748
749             constraints.fill = GridBagConstraints.BOTH;
750             constraints.weightx = 0;
751             constraints.weighty = 0;
752             constraints.gridx = xIdx;
753             constraints.gridy = 0;
754             constraints.gridwidth = 1;
755             constraints.gridheight = 1;
756             JPanel dummy = new JPanel(new BorderLayout());
757             columnLayout.setConstraints(dummy, constraints);
758             dummy.add(text, BorderLayout.CENTER);
759             ret.add(dummy);
760
761             constraints.fill = GridBagConstraints.BOTH;
762             constraints.weightx = 100;
763             constraints.weighty = 100;
764             constraints.gridx = xIdx;
765             constraints.gridy = 1;
766             constraints.gridwidth = 1;
767             constraints.gridheight = 1;
768             dummy = new JPanel(new BorderLayout());
769             columnLayout.setConstraints(dummy, constraints);
770             dummy.add(child, BorderLayout.CENTER);
771             ret.add(dummy);
772
773             xIdx++;
774             if (prev != null) {
775                 prev.addColumnAfter(text);
776             }
777             prev = text;
778         }
779
780         return ret;
781     }
```

```

779     }
780
781     private JPanel getAutoGraphics(TypeAuto type, boolean isOutput, IntReference ref,
782                                     IntReference sref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
783     {
784         JPanel ret = new JPanel(new BorderLayout());
785         ConnectTextArea area = getTextArea(1,2,"Auto",type,isOutput,sref,typeConn);
786         area.setBackground(new Color(0xFF, 0xCC, 0xCC));
787         area.index = sref.value;
788         ret.add(area, BorderLayout.LINE_START);
789         sref.value += 1;
790         ret.add(getTypeGraphics(type.getType(),isOutput,ref,sref,connectAreas,typeConn),
791                 BorderLayout.LINE_END);
792         return ret;
793     }
794
795     private JPanel getListGraphics(TypeList type, boolean isOutput, IntReference ref,
796                                    IntReference sref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
797     {
798         JPanel ret = new JPanel(new BorderLayout());
799         ConnectTextArea area = getTextArea(1,2,"List",type,isOutput,sref,typeConn);
800         area.setBackground(new Color(0xCC, 0xFF, 0xCC));
801         area.index = sref.value;
802         ret.add(area, BorderLayout.LINE_START);
803         sref.value += 1;
804         ret.add(getTypeGraphics(type.getChildType(),isOutput,ref,sref,connectAreas,typeConn),
805                 BorderLayout.LINE_END);
806         return ret;
807     }
808
809     private JPanel getLeafTypeGraphics(String name, lang.type.Type type, boolean isOutput,
810                                       IntReference ref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
811     {
812         JPanel ret = new JPanel(new BorderLayout());
813         JTextArea area = getTextArea(1,5,name,type,isOutput,ref,typeConn);
814         area.setBackground(new Color(0xd3, 0xd3, 0xd3));
815         ret.add(area);
816         connectAreas.add((PrimitiveTextArea)area);
817         ref.value += 1;
818         return ret;
819     }
820
821     private JPanel getErrorGraphics(TypeError type, boolean isOutput, IntReference ref,
822                                    IntReference sref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
823     {
824         return getLeafTypeGraphics("Error",type,isOutput,ref,connectAreas,typeConn);
825     }
826
827     private JPanel getUnknownGraphics(TypeUnknown type, boolean isOutput, IntReference ref,
828                                      IntReference sref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
829     {
830         return getLeafTypeGraphics("Unknown",type,isOutput,ref,connectAreas,typeConn);
831     }
832
833     private JPanel getTimeGraphics(TypeTime type, boolean isOutput, IntReference ref,
834                                    IntReference sref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
835     {
836         JPanel ret = new JPanel(new BorderLayout());
837         ConnectTextArea area = getTextArea(1,2,"Time",type,isOutput,sref,typeConn);
838         area.setBackground(new Color(0x99, 0x66, 0x33));
839         area.index = sref.value;
840         ret.add(area, BorderLayout.LINE_START);
841         sref.value += 1;
842         ret.add(getTypeGraphics(type.getType(),isOutput,ref,sref,connectAreas,typeConn),
843                 BorderLayout.LINE_END);
844         return ret;
845     }
846
847     private JPanel getBooleanGraphics(TypeBoolean type, boolean isOutput, IntReference ref,
848                                       IntReference sref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
849     {
850         JPanel ret = new JPanel(new BorderLayout());
851         ConnectTextArea area = getTextArea(1,2,"Boolean",type,isOutput,sref,typeConn);
852         area.setBackground(new Color(0x33, 0x99, 0x33));
853         area.index = sref.value;
854         ret.add(area, BorderLayout.LINE_START);
855         sref.value += 1;
856         ret.add(getTypeGraphics(type.getType(),isOutput,ref,sref,connectAreas,typeConn),
857                 BorderLayout.LINE_END);
858         return ret;
859     }
860
861     private JPanel getNumberGraphics(TypeNumber type, boolean isOutput, IntReference ref,
862                                     IntReference sref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
863     {
864         JPanel ret = new JPanel(new BorderLayout());
865         ConnectTextArea area = getTextArea(1,2,"Number",type,isOutput,sref,typeConn);
866         area.setBackground(new Color(0x66, 0x99, 0x66));
867         area.index = sref.value;
868         ret.add(area, BorderLayout.LINE_START);
869         sref.value += 1;
870         ret.add(getTypeGraphics(type.getType(),isOutput,ref,sref,connectAreas,typeConn),
871                 BorderLayout.LINE_END);
872         return ret;
873     }
874
875     private JPanel getStringGraphics(TypeString type, boolean isOutput, IntReference ref,
876                                     IntReference sref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
877     {
878         JPanel ret = new JPanel(new BorderLayout());
879         ConnectTextArea area = getTextArea(1,2,"String",type,isOutput,sref,typeConn);
880         area.setBackground(new Color(0x99, 0x66, 0x99));
881         area.index = sref.value;
882         ret.add(area, BorderLayout.LINE_START);
883         sref.value += 1;
884         ret.add(getTypeGraphics(type.getType(),isOutput,ref,sref,connectAreas,typeConn),
885                 BorderLayout.LINE_END);
886         return ret;
887     }
888
889     private JPanel getBooleanListGraphics(TypeBooleanList type, boolean isOutput, IntReference ref,
890                                         IntReference sref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
891     {
892         JPanel ret = new JPanel(new BorderLayout());
893         ConnectTextArea area = getTextArea(1,2,"BooleanList",type,isOutput,sref,typeConn);
894         area.setBackground(new Color(0x33, 0x99, 0x33));
895         area.index = sref.value;
896         ret.add(area, BorderLayout.LINE_START);
897         sref.value += 1;
898         ret.add(getTypeGraphics(type.getType(),isOutput,ref,sref,connectAreas,typeConn),
899                 BorderLayout.LINE_END);
900         return ret;
901     }
902
903     private JPanel getNumberListGraphics(TypeNumberList type, boolean isOutput, IntReference ref,
904                                         IntReference sref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
905     {
906         JPanel ret = new JPanel(new BorderLayout());
907         ConnectTextArea area = getTextArea(1,2,"NumberList",type,isOutput,sref,typeConn);
908         area.setBackground(new Color(0x66, 0x99, 0x66));
909         area.index = sref.value;
910         ret.add(area, BorderLayout.LINE_START);
911         sref.value += 1;
912         ret.add(getTypeGraphics(type.getType(),isOutput,ref,sref,connectAreas,typeConn),
913                 BorderLayout.LINE_END);
914         return ret;
915     }
916
917     private JPanel getStringListGraphics(TypeStringList type, boolean isOutput, IntReference ref,
918                                         IntReference sref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
919     {
920         JPanel ret = new JPanel(new BorderLayout());
921         ConnectTextArea area = getTextArea(1,2,"StringList",type,isOutput,sref,typeConn);
922         area.setBackground(new Color(0x99, 0x66, 0x99));
923         area.index = sref.value;
924         ret.add(area, BorderLayout.LINE_START);
925         sref.value += 1;
926         ret.add(getTypeGraphics(type.getType(),isOutput,ref,sref,connectAreas,typeConn),
927                 BorderLayout.LINE_END);
928         return ret;
929     }
930
931     private JPanel getBooleanMapGraphics(TypeBooleanMap type, boolean isOutput, IntReference ref,
932                                         IntReference sref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
933     {
934         JPanel ret = new JPanel(new BorderLayout());
935         ConnectTextArea area = getTextArea(1,2,"BooleanMap",type,isOutput,sref,typeConn);
936         area.setBackground(new Color(0x33, 0x99, 0x33));
937         area.index = sref.value;
938         ret.add(area, BorderLayout.LINE_START);
939         sref.value += 1;
940         ret.add(getTypeGraphics(type.getType(),isOutput,ref,sref,connectAreas,typeConn),
941                 BorderLayout.LINE_END);
942         return ret;
943     }
944
945     private JPanel getNumberMapGraphics(TypeNumberMap type, boolean isOutput, IntReference ref,
946                                         IntReference sref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
947     {
948         JPanel ret = new JPanel(new BorderLayout());
949         ConnectTextArea area = getTextArea(1,2,"NumberMap",type,isOutput,sref,typeConn);
950         area.setBackground(new Color(0x66, 0x99, 0x66));
951         area.index = sref.value;
952         ret.add(area, BorderLayout.LINE_START);
953         sref.value += 1;
954         ret.add(getTypeGraphics(type.getType(),isOutput,ref,sref,connectAreas,typeConn),
955                 BorderLayout.LINE_END);
956         return ret;
957     }
958
959     private JPanel getStringMapGraphics(TypeStringMap type, boolean isOutput, IntReference ref,
960                                         IntReference sref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
961     {
962         JPanel ret = new JPanel(new BorderLayout());
963         ConnectTextArea area = getTextArea(1,2,"StringMap",type,isOutput,sref,typeConn);
964         area.setBackground(new Color(0x99, 0x66, 0x99));
965         area.index = sref.value;
966         ret.add(area, BorderLayout.LINE_START);
967         sref.value += 1;
968         ret.add(getTypeGraphics(type.getType(),isOutput,ref,sref,connectAreas,typeConn),
969                 BorderLayout.LINE_END);
970         return ret;
971     }
972
973     private JPanel getBooleanSetGraphics(TypeBooleanSet type, boolean isOutput, IntReference ref,
974                                         IntReference sref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
975     {
976         JPanel ret = new JPanel(new BorderLayout());
977         ConnectTextArea area = getTextArea(1,2,"BooleanSet",type,isOutput,sref,typeConn);
978         area.setBackground(new Color(0x33, 0x99, 0x33));
979         area.index = sref.value;
980         ret.add(area, BorderLayout.LINE_START);
981         sref.value += 1;
982         ret.add(getTypeGraphics(type.getType(),isOutput,ref,sref,connectAreas,typeConn),
983                 BorderLayout.LINE_END);
984         return ret;
985     }
986
987     private JPanel getNumberSetGraphics(TypeNumberSet type, boolean isOutput, IntReference ref,
988                                         IntReference sref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
989     {
990         JPanel ret = new JPanel(new BorderLayout());
991         ConnectTextArea area = getTextArea(1,2,"NumberSet",type,isOutput,sref,typeConn);
992         area.setBackground(new Color(0x66, 0x99, 0x66));
993         area.index = sref.value;
994         ret.add(area, BorderLayout.LINE_START);
995         sref.value += 1;
996         ret.add(getTypeGraphics(type.getType(),isOutput,ref,sref,connectAreas,typeConn),
997                 BorderLayout.LINE_END);
998         return ret;
999     }
999
999 
```

```

827         IntReference sref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
828     {
829         return getLeafTypeGraphics("Time",type,isOutput,ref,connectAreas,typeConn);
830     }
831
832     private JPanel getStringGraphics(TypeString type, boolean isOutput, IntReference ref,
833         IntReference sref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
834     {
835         return getLeafTypeGraphics("String",type,isOutput,ref,connectAreas,typeConn);
836     }
837
838     private JPanel getBoolGraphics(TypeBool b, boolean isOutput, IntReference ref,
839         IntReference sref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
840     {
841         return getLeafTypeGraphics("Boolean",b,isOutput,ref,connectAreas,typeConn);
842     }
843
844     private JPanel getNumberGraphics(TypeNumber n, boolean isOutput, IntReference ref,
845         IntReference sref, ArrayList<PrimitiveTextArea> connectAreas, TypeConnection typeConn)
846     {
847         return getLeafTypeGraphics("Number",n,isOutput,ref,connectAreas,typeConn);
848     }
849 }
```

LISTING E.101: gui/TypeConnectionPanel.java

```

1 package gui;
2
3 import gui.TabbedPane.InvalidTitleException;
4 import gui.controlflow.ControlArrow;
5 import gui.controlflow.ControlPart;
6 import gui.controlflow.ControlPartTypeConnection;
7 import gui.dataflow.DataPart;
8 import gui.dataflow.DataPartComparison.ComparisonType;
9 import gui.dataflow.DataPartFunctional.FunctionalType;
10 import gui.dataflow.DataPartListOperation.ListOperationType;
11 import gui.dataflow.DataPartFunctional;
12 import gui.dataflow.DataPartParameter;
13 import gui.dataflow.DataPartResult;
14 import gui.dataflow.DataPartStringOperation.StringOperationType;
15 import gui.dataflow.DataPartSubFlow;
16 import gui.dataflow.GraphUtil;
17 import gui.dataflow.IDataPartSubFlow;
18 import gui.dataflow.InvalidDataPartException;
19 import gui.dataflow.DataPartArithmetic.ArithmeticType;
20
21 import java.awt.Point;
22 import java.io.File;
23 import java.io.IOException;
24 import java.math.BigDecimal;
25 import java.util.ArrayList;
26 import java.util.Collections;
27 import java.util.HashMap;
28 import java.util.List;
29 import java.util.Map;
```

```
30 import java.util.Stack;
31
32 import javax.xml.parsers.ParserConfigurationException;
33 import javax.xml.parsers.SAXParser;
34 import javax.xml.parsers.SAXParserFactory;
35
36 import lang.constant.Constant;
37 import lang.constant.ConstantBool;
38 import lang.constant.ConstantList;
39 import lang.constant.ConstantNumber;
40 import lang.constant.ConstantRecord;
41 import lang.constant.ConstantString;
42 import lang.constant.RecordValue;
43 import lang.type.RecordColumn;
44 import lang.type.Type;
45 import lang.type.TypeAuto;
46 import lang.type.TypeBool;
47 import lang.type.TypeConnection;
48 import lang.type.TypeError;
49 import lang.type.TypeList;
50 import lang.type.TypeNumber;
51 import lang.type.TypeRecord;
52 import lang.type.TypeString;
53 import lang.type.TypeTime;
54 import lang.type.TypeUnknown;
55
56 import org.apache.commons.lang3.StringEscapeUtils;
57 import org.xml.sax.Attributes;
58 import org.xml.sax.SAXException;
59 import org.xml.sax.helpers.DefaultHandler;
60
61 public class ASTLoader extends DefaultHandler {
62     private MainWindow mainWindow;
63     private ControlFlowPanel rootControlFlow;
64     private ArrayList<ControlPartSeq> controlPartSeq = new ArrayList<>();
65     private Stack<DataFlowEnvironment> dataFlowPanelStack = new Stack<>();
66     private Map<String,ControlPart> controlPartMap = new HashMap<>();
67     private ControlArrow currentControlArrow;
68     private DataPart.InputArrow currentDataArrow;
69     private TypeConnection currentTypeConnection;
70     private TypeEvent nextTypeEvent;
71     private ConstantEvent nextConstantEvent;
72     private Stack<TypeRecordBuilder> typeRecordBuilderStack = new Stack<>();
73     private Stack<ConstantRecordBuilder> constantRecordBuilderStack = new Stack<>();
74     private Stack<ConstantListBuilder> constantListBuilderStack = new Stack<>();
75     private CastEnvironment castEnvironment;
76     private FunctionalEnvironment functionalEvnironment;
77     private InteractionEnvironment interactionEnvironment;
78     private ListOperationEnvironment listOperationEnvironment;
79     private RecordConstructorEnvironment recordConstructorEnvironment;
80     private ComparisonEnvironment comparisonEnvironment;
81     private IOEnvironment ioEnvironment;
82
83     private static class ControlPartSeq {
84         int seqNumber;
```

```
85     ControlPart part;
86     public ControlPartSeq(int n, ControlPart p) {
87         seqNumber = n;
88         part = p;
89     }
90 }
91
92     private static class DataPartSeq {
93         int seqNumber;
94         DataPart part;
95         public DataPartSeq(int n, DataPart p) {
96             seqNumber = n;
97             part = p;
98         }
99     }
100
101    private static class IOEnvironment {
102
103        public String id;
104        public Point location;
105        public String name;
106        public int seq;
107        public Type type;
108        public Type originalType;
109
110    }
111
112    private static class ListOperationEnvironment {
113
114        Point location;
115        int seq;
116        String id;
117        ListOperationType type;
118        List<Type> opTypes = new ArrayList<>();
119        List<Type> originalOpTypes = new ArrayList<>();
120
121    }
122
123    private static class ComparisonEnvironment {
124
125        String id;
126        Point location;
127        ComparisonType compType;
128        int seq;
129        Type dataType;
130        Type originalDataType;
131
132    }
133
134    private static class InteractionEnvironment {
135
136        String title;
137        List<String> events;
138        List<RecordColumn> inputs;
139        List<RecordColumn> outputs;
140        int seq;
141        Point location;
142        String id;
143        String platform;
144
145    }
```

```
140
141     private static class CastEnvironment {
142         Point location;
143         String id;
144         int seq;
145         public Type fromType;
146         public Type toType;
147         public Type originalFromType;
148         public CastEnvironment(Point loc, String id, int seq) {
149             location = loc;
150             this.id = id;
151             this.seq = seq;
152         }
153     }
154
155     private static class DataFlowEnvironment {
156         List<DataPartSeq> components = new ArrayList<>();
157         DataFlowPanel panel;
158         Map<String,DataPart> dataPartMap;
159         boolean isFunctional;
160         DataFlowEnvironment(DataFlowPanel panel, boolean func) {
161             this.panel = panel;
162             dataPartMap = new HashMap<>();
163             isFunctional = func;
164         }
165     }
166
167     private static class FunctionalEnvironment {
168         String id;
169         String subId;
170         FunctionalType funType;
171         int seq;
172         Point location;
173         List<RecordColumn> inputTypes = new ArrayList<>();
174         List<RecordColumn> outputTypes = new ArrayList<>();
175     }
176
177     private class RecordConstructorEnvironment {
178         String id;
179         int seq;
180         Point location;
181         TypeRecord type;
182     }
183
184     private class ConstantRecordBuilder {
185         ArrayList<RecordValue> columns = new ArrayList<>();
186         ConstantEvent finalEvent;
187         public ConstantRecordBuilder(ConstantEvent event) {
188             finalEvent = event;
189         }
190     }
191
192     private class ConstantListBuilder {
193         ArrayList<Constant> values = new ArrayList<>();
194         ConstantEvent finalEvent;
```

```
195     public Type type;
196     public ConstantListBuilder(ConstantEvent event) {
197         finalEvent = event;
198     }
199 }
200
201     private class TypeRecordBuilder {
202         ArrayList<RecordColumn> columns = new ArrayList<>();
203         TypeEvent finalEvent;
204         public TypeRecordBuilder(TypeEvent event) {
205             finalEvent = event;
206         }
207     }
208
209     @FunctionalInterface
210     private interface ConstantEvent {
211         public void apply(Constant c) throws InvalidTitleException, InvalidDataPartException;
212     }
213
214     @FunctionalInterface
215     private interface TypeEvent {
216         public void apply(Type t) throws InvalidTitleException, InvalidDataPartException;
217     }
218
219     public ASTLoader(MainWindow mainWin) {
220         mainWindow = mainWin;
221     }
222
223     private Point getLocation(Attributes attrs) {
224         int x = Integer.parseInt(attrs.getValue("x"));
225         int y = Integer.parseInt(attrs.getValue("y"));
226         return new Point(x,y);
227     }
228
229     private void startControlFlow(Attributes attrs)
230         throws InvalidTitleException {
231         rootControlFlow = new ControlFlowPanel(mainWindow, attrs.getValue("id"), attrs.getValue("platform"),
232             attrs.getValue("group"), null);
233         mainWindow.addExplorerItem(null, rootControlFlow.getScrollPane());
234     }
235
236     private void endControlFlow() {
237
238     private void startComponents(Attributes attrs) {}
239     private void endComponents() {}
240
241     private void startStart(Attributes attrs) {
242         ControlPart cp = rootControlFlow.getStartControlPart();
243         Point p = getLocation(attrs);
244         cp.setLocation(p);
245         controlPartMap.put(attrs.getValue("id"), cp);
246     }
247     private void endStart() {}
```

```
249     private void startStop(Attributes attrs) {
250         ControlPart p = ControlPart.getStopControlPart(rootControlFlow, mainWindow.getGraphics());
251         p.setLocation(getLocation(attrs));
252         controlPartMap.put(attrs.getValue("id"), p);
253         int seq = Integer.parseInt(attrs.getValue("seq"));
254         controlPartSeq.add(new ControlPartSeq(seq, p));
255     }
256     private void endStop() {}
257
258     private void startFail(Attributes attrs) {
259         ControlPart p = ControlPart.getFailControlPart(rootControlFlow, mainWindow.getGraphics());
260         p.setLocation(getLocation(attrs));
261         controlPartMap.put(attrs.getValue("id"), p);
262         int seq = Integer.parseInt(attrs.getValue("seq"));
263         controlPartSeq.add(new ControlPartSeq(seq, p));
264     }
265     private void endFail() {}
266
267     private void startIf(Attributes attrs) {
268         ControlPart p = ControlPart.getIfRuntimeInteractionControlPart(rootControlFlow, mainWindow
269             .getGraphics());
270         p.setLocation(getLocation(attrs));
271         controlPartMap.put(attrs.getValue("id"), p);
272         int seq = Integer.parseInt(attrs.getValue("seq"));
273         controlPartSeq.add(new ControlPartSeq(seq, p));
274     }
275     private void endIf() {}
276
277     private void startInteractionComponent(Attributes attrs) {
278         interactionEnvironment = new InteractionEnvironment();
279         interactionEnvironment.seq = Integer.parseInt(attrs.getValue("seq"));
280         interactionEnvironment.title = attrs.getValue("title");
281         interactionEnvironment.location = getLocation(attrs);
282         interactionEnvironment.id = attrs.getValue("id");
283         interactionEnvironment.platform = attrs.getValue("platform");
284     }
285     private void endInteractionComponent() {
286         String title = interactionEnvironment.title;
287         String[] events = new String[interactionEnvironment.events.size()];
288         String platform = interactionEnvironment.platform;
289         for (int i = 0; i < events.length; i++)
290             events[i] = interactionEnvironment.events.get(i);
291         List<RecordColumn> inputs = interactionEnvironment.inputs;
292         List<RecordColumn> outputs = interactionEnvironment.outputs;
293         ControlPart p = ControlPart.getInteractionControlPart(title,
294             platform, events, inputs, outputs, rootControlFlow, mainWindow.getGraphics());
295         p.setLocation(interactionEnvironment.location);
296         controlPartMap.put(interactionEnvironment.id, p);
297         int seq = interactionEnvironment.seq;
298         controlPartSeq.add(new ControlPartSeq(seq, p));
299         interactionEnvironment = null;
300     }
301
302     private void startInteractionInputs(Attributes attrs) {
303         interactionEnvironment.inputs = new ArrayList<>();
```

```
303     }
304     private void endInteractionInputs(){}
305
306     private void startInteractionOutputs(Attributes attrs) {
307         interactionEnvironment.outputs = new ArrayList<>();
308     }
309     private void endInteractionOutputs() {}
310
311     private void startInteractionInput(Attributes attrs) {
312         String id = attrs.getValue("id");
313         nextTypeEvent = (t) -> {
314             interactionEnvironment.inputs.add(new RecordColumn(id, t));
315         };
316     }
317     private void endInteractionInput() {
318         nextTypeEvent = null;
319     }
320
321     private void startInteractionOutput(Attributes attrs) {
322         String id = attrs.getValue("id");
323         nextTypeEvent = (t) -> {
324             interactionEnvironment.outputs.add(new RecordColumn(id, t));
325         };
326     }
327     private void endInteractionOutput() {
328         nextTypeEvent = null;
329     }
330
331     private void startInteractionEvents(Attributes attrs) {
332         interactionEnvironment.events = new ArrayList<>();
333     }
334     private void endInteractionEvents() {}
335
336     private void startInteractionEvent(Attributes attrs) {
337         interactionEnvironment.events.add(attrs.getValue("id"));
338     }
339     private void endInteractionEvent() {}
340
341     private void startDataFlowComponent(Attributes attrs)
342         throws InvalidTitleException, InvalidDataPartException {
343         if (dataFlowPanelStack.size() == 0) {
344             ControlPart p = ControlPart.getDataflowControlPart(mainWindow, attrs.getValue("id"),
345             rootControlFlow, mainWindow.getGraphics());
346             mainWindow.addExplorerItem(rootControlFlow, p.getTabbedPanel());
347             p.setLocation(getLocation(attrs));
348             controlPartMap.put(attrs.getValue("id"), p);
349             int seq = Integer.parseInt(attrs.getValue("seq"));
350             controlPartSeq.add(new ControlPartSeq(seq, p));
351             DataFlowPanel dataFlowPanel = (DataFlowPanel)p.getTabbedPanel();
352             dataFlowPanelStack.push(new DataFlowEnvironment(dataFlowPanel, false));
353         } else {
354             DataFlowEnvironment env = dataFlowPanelStack.lastElement();
355             DataPartSubFlow p = env.panel.getNewDataPartSubFlow(attrs.getValue("id"));
356             p.setPointBoxLocation(getLocation(attrs));
357             env.dataPartMap.put(attrs.getValue("id"), p);
```

```
357         env.components.add(new DataPartSeq(Integer.parseInt(attrs.getValue("seq")), p));
358         DataFlowPanel dataFlowPanel = ((IDataPartSubFlow)p).getSubDataFlowPanel();
359         dataFlowPanelStack.push(new DataFlowEnvironment(dataFlowPanel, false));
360     }
361 }
362 private void endDataFlowComponent() {
363     dataFlowPanelStack.pop();
364 }
365
366 private void startEdges(Attributes attrs) {
367     if (dataFlowPanelStack.size() == 0) {
368         ArrayList<ControlPartSeq> sorted = new ArrayList<>(controlPartSeq);
369         Collections.sort(sorted, (xx,yy) -> {
370             if (xx.seqNumber < yy.seqNumber)
371                 return -1;
372             if (xx.seqNumber == yy.seqNumber)
373                 return 0;
374             return 1;
375         });
376         for (ControlPartSeq p : sorted) {
377             rootControlFlow.addControlPart(p.part, false);
378         }
379
380         List<ControlPart> parts = new ArrayList<>();
381         for (ControlPartSeq p : controlPartSeq) {
382             parts.add(p.part);
383         }
384         rootControlFlow.setNonStartControlParts(parts);
385         rootControlFlow.resetCanFail();
386     } else {
387         DataFlowEnvironment env = dataFlowPanelStack.lastElement();
388
389         ArrayList<DataPartSeq> sorted = new ArrayList<>(env.components);
390         Collections.sort(sorted, (xx,yy) -> {
391             if (xx.seqNumber < yy.seqNumber)
392                 return -1;
393             if (xx.seqNumber == yy.seqNumber)
394                 return 0;
395             return 1;
396         });
397         for (DataPartSeq p : sorted) {
398             if (env.isFunctional) {
399                 if (p.part instanceof DataPartParameter)
400                     continue;
401                 if (p.part instanceof DataPartResult)
402                     continue;
403             }
404             env.panel.addDataPart(p.part, false);
405         }
406
407         List<DataPart> parts = new ArrayList<>();
408         for (DataPartSeq p : env.components) {
409             parts.add(p.part);
410         }
411         env.panel.setDataPartsClearTemps(parts);
```

```
412     }
413 }
414 private void endEdges() {}
415
416 private void startEdge(Attributes attrs) {
417     if (dataFlowPanelStack.size() == 0) {
418         int eventIndex = Integer.parseInt(attrs.getValue("eventIndex"));
419         ControlPart start = controlPartMap.get(attrs.getValue("start"));
420         ControlPart end = controlPartMap.get(attrs.getValue("end"));
421         ControlArrow arrow = new ControlArrow(start, eventIndex, end);
422         currentControlArrow = arrow;
423         rootControlFlow.addControlArrow(arrow);
424     } else {
425         DataFlowEnvironment env = dataFlowPanelStack.lastElement();
426         String outputName = attrs.getValue("outputName");
427         DataPart outputPart = env.dataPartMap.get(attrs.getValue("outputComponent"));
428         String inputName = attrs.getValue("inputName");
429         DataPart inputPart = env.dataPartMap.get(attrs.getValue("inputComponent"));
430
431         DataPart.InputArrow inA = inputPart.getInputArrow(inputName);
432         DataPart.OutputArrow outA = outputPart.getOutputArrow(outputName);
433         GraphUtil.connectArrows(inA, outA);
434         currentTypeConnection = new TypeConnection(outA.getType(), inA.getType(), false);
435         currentDataArrow = inA;
436     }
437 }
438 private void endEdge() {
439     currentControlArrow = null;
440     if (dataFlowPanelStack.size() > 0) {
441         currentDataArrow.setTypeConnection(currentTypeConnection);
442         currentTypeConnection = null;
443     }
444 }
445
446 private void startTypeConnection(Attributes attrs) {
447     String outName = attrs.getValue("outputName");
448     ControlPart outPart = controlPartMap.get(attrs.getValue("outputComponent"));
449     String inName = attrs.getValue("inputName");
450     ControlPart inPart = controlPartMap.get(attrs.getValue("inputComponent"));
451     ControlPartTypeConnection conn = new ControlPartTypeConnection(outPart, outName, inPart,
452     inName, false);
453     currentTypeConnection = conn;
454     currentControlArrow.putArgumentConnection(conn);
455 }
456 private void endTypeConnection() {
457     currentTypeConnection = null;
458 }
459
460 private void startConnection(Attributes attrs) {
461     int outIndex = Integer.parseInt(attrs.getValue("outputIndex"));
462     int inIndex = Integer.parseInt(attrs.getValue("inputIndex"));
463     currentTypeConnection.addConnection(new TypeConnection.SingleConnection(outIndex, inIndex)
464 );
465 }
466 private void endConnection() {}
```

```
465
466     private void startDataFlow(Attributes attrs) {}
467     private void endDataFlow() {}
468
469     private void startSinkComponent(Attributes attrs) {
470         Point location = getLocation(attrs);
471         int seq = Integer.parseInt(attrs.getValue("seq"));
472         String id = attrs.getValue("id");
473
474         DataFlowEnvironment env = dataFlowPanelStack.lastElement();
475
476         nextTypeEvent = (type) -> {
477             DataPart p = env.panel.getNewDataPartSink(type);
478             env.dataPartMap.put(id, p);
479             p.setPointBoxLocation(location);
480             env.components.add(new DataPartSeq(seq, p));
481         };
482     }
483     private void endSinkComponent() {
484         nextTypeEvent = null;
485     }
486
487     private void startInputComponent(Attributes attrs) {
488         startOutputComponent(attrs);
489     }
490     private void endInputComponent() throws InvalidDataPartException {
491         DataFlowEnvironment env = dataFlowPanelStack.lastElement();
492         DataFlowPanel panel = env.panel;
493
494         String id = ioEnvironment.id;
495         Point location = ioEnvironment.location;
496         String name = ioEnvironment.name;
497         int seq = ioEnvironment.seq;
498         Type type = ioEnvironment.type;
499         Type originalType = ioEnvironment.originalType;
500
501         if (env.isFunctional) {
502             DataPartParameter p = env.panel.getDataPartParameter(name);
503             env.dataPartMap.put(id, p);
504             p.setPointBoxLocation(location);
505             env.components.add(new DataPartSeq(seq, p));
506         } else {
507             DataPart p = panel.getNewDataPartParameter(name, originalType, type,
508                 panel.getParentTabbedPane().getChildParamaterDeleteEvent(), true);
509             env.dataPartMap.put(id, p);
510             p.setPointBoxLocation(location);
511             env.components.add(new DataPartSeq(seq, p));
512         }
513     }
514
515     private void startInputType(Attributes attrs) {
516         startOutputType(attrs);
517     }
518     private void endInputType() {
519         nextTypeEvent = null;
```

```
520     }
521
522     private void startInputOriginalType(Attributes attrs) {
523         startOutputOriginalType(attrs);
524     }
525     private void endInputOriginalType() {
526         nextTypeEvent = null;
527     }
528
529     private void startOutputComponent(Attributes attrs) {
530         ioEnvironment = new IOEnvironment();
531
532         String id = attrs.getValue("id");
533         Point location = getLocation(attrs);
534         assert id != null;
535         String name = attrs.getValue("name");
536         int seq = Integer.parseInt(attrs.getValue("seq"));
537
538         ioEnvironment.id = id;
539         ioEnvironment.location = location;
540         ioEnvironment.name = name;
541         ioEnvironment.seq = seq;
542     }
543
544     private void endOutputComponent() throws InvalidDataPartException {
545         DataFlowEnvironment env = dataFlowPanelStack.lastElement();
546         DataFlowPanel panel = env.panel;
547
548         String id = ioEnvironment.id;
549         Point location = ioEnvironment.location;
550         String name = ioEnvironment.name;
551         int seq = ioEnvironment.seq;
552         Type outputType = ioEnvironment.type;
553         Type outputOriginalType = ioEnvironment.originalType;
554
555         if (env.isFunctional) {
556             DataPartResult p = env.panel.getDataPartResult(name);
557             env.dataPartMap.put(id, p);
558             p.setPointBoxLocation(location);
559             env.components.add(new DataPartSeq(seq, p));
560         } else {
561             DataPart p = panel.getNewDataPartResult(name, outputOriginalType, outputType,
562                 panel.getParentTabbedPane().getChildResultDeleteEvent(), true);
563             env.dataPartMap.put(id, p);
564             p.setPointBoxLocation(location);
565             env.components.add(new DataPartSeq(seq, p));
566         }
567
568         nextTypeEvent = null;
569     }
570
571     private void startOutputType(Attributes attrs) {
572         DataFlowEnvironment env = dataFlowPanelStack.lastElement();
573         if (env.isFunctional) {
574             nextTypeEvent = (type) -> {};
```

```
575     } else {
576         nextTypeEvent = (type) -> {
577             ioEnvironment.type = type;
578         };
579     }
580 }
581 private void endOutputType() {
582     nextTypeEvent = null;
583 }
584
585 private void startOutputOriginalType(Attributes attrs) {
586     DataFlowEnvironment env = dataFlowPanelStack.lastElement();
587     if (env.isFunctional) {
588         nextTypeEvent = (type) -> {};
589     } else {
590         nextTypeEvent = (type) -> {
591             ioEnvironment.originalType = type;
592         };
593     }
594 }
595 private void endOutputOriginalType() {
596     nextTypeEvent = null;
597 }
598
599 private void startComparisonComponent(Attributes attrs) {
600     String id = attrs.getValue("id");
601     Point location = getLocation(attrs);
602     assert id != null;
603     ComparisonType compType = ComparisonType.valueOf(attrs.getValue("type"));
604     int seq = Integer.parseInt(attrs.getValue("seq"));
605
606     comparisonEnvironment = new ComparisonEnvironment();
607     comparisonEnvironment.id = id;
608     comparisonEnvironment.location = location;
609     comparisonEnvironment.compType = compType;
610     comparisonEnvironment.seq = seq;
611 }
612 private void endComparisonComponent() throws InvalidDataPartException {
613     String id = comparisonEnvironment.id;
614     Point location = comparisonEnvironment.location;
615     int seq = comparisonEnvironment.seq;
616     ComparisonType compType = comparisonEnvironment.compType;
617
618     Type dataType = comparisonEnvironment.dataType;
619     Type originalDataType = comparisonEnvironment.originalDataType;
620
621     DataFlowEnvironment env = dataFlowPanelStack.lastElement();
622     DataFlowPanel panel = env.panel;
623     DataPart p = panel.getNewDataPartComparison(compType, originalDataType, dataType);
624     env.dataPartMap.put(id, p);
625     p.setPointBoxLocation(location);
626     env.components.add(new DataPartSeq(seq, p));
627
628     comparisonEnvironment = null;
629 }
```

```
630
631     private void startComparisonDataType(Attributes attrs) {
632         nextTypeEvent = (type) -> {
633             comparisonEnvironment.dataType = type;
634         };
635     }
636
637     private void endComparisonDataType() {
638         nextTypeEvent = null;
639     }
640
641     private void startComparisonOriginalDataType(Attributes attrs) {
642         nextTypeEvent = (type) -> {
643             comparisonEnvironment.originalDataType = type;
644         };
645     }
646
647     private void endComparisonOriginalDataType() {
648         nextTypeEvent = null;
649     }
650
651     private void startArithmeticComponent(Attributes attrs) {
652         DataFlowEnvironment env = dataFlowPanelStack.lastElement();
653         DataFlowPanel panel = env.panel;
654         String id = attrs.getValue("id");
655         Point location = getLocation(attrs);
656         assert id != null;
657         ArithmeticType arithType = ArithmeticType.valueOf(attrs.getValue("type"));
658         int seq = Integer.parseInt(attrs.getValue("seq"));
659
660         DataPart p = panel.getNewDataPartArithmetic(arithType);
661         env.dataPartMap.put(id, p);
662         p.setPointBoxLocation(location);
663         env.components.add(new DataPartSeq(seq, p));
664     }
665     private void endArithmeticComponent() {}
666
667     private void startStringOperationComponent(Attributes attrs) {
668         DataFlowEnvironment env = dataFlowPanelStack.lastElement();
669         DataFlowPanel panel = env.panel;
670         String id = attrs.getValue("id");
671         Point location = getLocation(attrs);
672         StringOperationType sType = StringOperationType.valueOf(attrs.getValue("type"));
673         int seq = Integer.parseInt(attrs.getValue("seq"));
674
675         DataPart p = panel.getNewDataPartStringOperation(sType);
676         env.dataPartMap.put(id, p);
677         p.setPointBoxLocation(location);
678         env.components.add(new DataPartSeq(seq, p));
679     }
680     private void endStringOperationComponent() {}
681
682     private void startCastComponent(Attributes attrs) {
683         castEnvironment = new CastEnvironment(getLocation(attrs), attrs.getValue("id"),
684             Integer.parseInt(attrs.getValue("seq")));
685     }
```

```
685     }
686     private void endCastComponent() throws InvalidDataPartException {
687         DataFlowEnvironment env = dataFlowPanelStack.lastElement();
688         DataFlowPanel panel = env.panel;
689
690         DataPart p = panel.getNewDataPartCast(castEnvironment.originalFromType,
691             castEnvironment.fromType, castEnvironment.toType);
692         env.dataPartMap.put(castEnvironment.id, p);
693         p.setPointBoxLocation(castEnvironment.location);
694         env.components.add(new DataPartSeq(castEnvironment.seq, p));
695
696         castEnvironment = null;
697     }
698
699     private void startOriginalCastFrom(Attributes attrs) {
700         nextTypeEvent = (type) -> {
701             castEnvironment.originalFromType = type;
702         };
703     }
704     private void endOriginalCastFrom() {
705         nextTypeEvent = null;
706     }
707
708     private void startCastFrom(Attributes attrs) {
709         nextTypeEvent = (type) -> {
710             castEnvironment.fromType = type;
711         };
712     }
713     private void endCastFrom() {
714         nextTypeEvent = null;
715     }
716
717     private void startCastTo(Attributes attrs) {
718         nextTypeEvent = (type) -> {
719             castEnvironment.toType = type;
720         };
721     }
722     private void endCastTo() {
723         nextTypeEvent = null;
724     }
725
726     private void startListOperationComponent(Attributes attrs) {
727         listOperationEnvironment = new ListOperationEnvironment();
728         listOperationEnvironment.location = getLocation(attrs);
729         listOperationEnvironment.seq = Integer.parseInt(attrs.getValue("seq"));
730         listOperationEnvironment.type = ListOperationType.valueOf(attrs.getValue("type"));
731         listOperationEnvironment.id = attrs.getValue("id");
732     }
733     private void endListOperationComponent() {
734         DataFlowEnvironment env = dataFlowPanelStack.lastElement();
735         DataPart p = env.panel.getNewDataPartListOperation(
736             listOperationEnvironment.type,
737             listOperationEnvironment.originalOpTypes,
738             listOperationEnvironment.opTypes);
739         env.dataPartMap.put(listOperationEnvironment.id, p);
```

```
740         p.setPointBoxLocation(listOperationEnvironment.location);
741         env.components.add(new DataPartSeq(listOperationEnvironment.seq, p));
742         listOperationEnvironment = null;
743         nextTypeEvent = null;
744     }
745
746     private void startListOperationType(Attributes attrs) {
747         nextTypeEvent = (t) -> {
748             listOperationEnvironment.opTypes.add(t);
749         };
750     }
751     private void endListOperationType() {
752         nextTypeEvent = null;
753     }
754
755     private void startListOperationOriginalType(Attributes attrs) {
756         nextTypeEvent = (t) -> {
757             listOperationEnvironment.originalOpTypes.add(t);
758         };
759     }
760     private void endListOperationOriginalType() {
761         nextTypeEvent = null;
762     }
763
764     private void startConstantComponent(Attributes attrs) {
765         DataFlowEnvironment env = dataFlowPanelStack.lastElement();
766         Point location = getLocation(attrs);
767         int seq = Integer.parseInt(attrs.getValue("seq"));
768         String id = attrs.getValue("id");
769
770         nextConstantEvent = (con) -> {
771             DataPart p = env.panel.getNewDataPartConstant(con);
772             env.dataPartMap.put(id, p);
773             p.setPointBoxLocation(location);
774             env.components.add(new DataPartSeq(seq, p));
775         };
776     }
777     private void endConstantComponent() {
778         nextConstantEvent = null;
779     }
780
781     private void startStringFormatComponent(Attributes attrs) throws InvalidDataPartException {
782         DataFlowEnvironment env = dataFlowPanelStack.lastElement();
783         Point location = getLocation(attrs);
784         int seq = Integer.parseInt(attrs.getValue("seq"));
785         String id = attrs.getValue("id");
786         String fmt = StringEscapeUtils.unescapeXml(attrs.getValue("fmt"));
787
788         DataPart p = env.panel.getNewDataPartStringFormat(fmt);
789         env.dataPartMap.put(id, p);
790         p.setPointBoxLocation(location);
791         env.components.add(new DataPartSeq(seq, p));
792     }
793     private void endStringFormatComponent() {}
```

```

795     private void startRecordConstructorComponent(Attributes attrs) {
796         Point location = getLocation(attrs);
797         int seq = Integer.parseInt(attrs.getValue("seq"));
798         String id = attrs.getValue("id");
799
800         recordConstructorEnvironment = new RecordConstructorEnvironment();
801         recordConstructorEnvironment.location = location;
802         recordConstructorEnvironment.id = id;
803         recordConstructorEnvironment.seq = seq;
804
805         nextTypeEvent = (t) -> {
806             recordConstructorEnvironment.type = (TypeRecord)t;
807         };
808     }
809
810     private void endRecordConstructorComponent() {
811         DataFlowEnvironment env = dataFlowPanelStack.lastElement();
812         Point location = recordConstructorEnvironment.location;
813         int seq = recordConstructorEnvironment.seq;
814         String id = recordConstructorEnvironment.id;
815         TypeRecord type = recordConstructorEnvironment.type;
816
817         DataPart p = env.panel.getNewDataPartRecordConstructor(type);
818         env.dataPartMap.put(id, p);
819         p.setPointBoxLocation(location);
820         env.components.add(new DataPartSeq(seq, p));
821
822         recordConstructorEnvironment = null;
823         nextTypeEvent = null;
824     }
825
826     private void startFunctionalComponent(Attributes attrs)
827         throws InvalidTitleException, InvalidDataPartException {
828         functionalEvnironment = new FunctionalEnvironment();
829         functionalEvnironment.id = attrs.getValue("id");
830         functionalEvnironment.subId = attrs.getValue("subId");
831         functionalEvnironment.location = getLocation(attrs);
832         functionalEvnironment.seq = Integer.parseInt(attrs.getValue("seq"));
833         functionalEvnironment.funType = FunctionalType.valueOf(attrs.getValue("type"));
834     }
835
836     private void endFunctionalComponent() {
837         functionalEvnironment = null;
838     }
839
840     private void startFunctionalSubDataFlow(Attributes attrs)
841         throws InvalidTitleException, InvalidDataPartException {
842         DataFlowEnvironment env = dataFlowPanelStack.lastElement();
843         String id = functionalEvnironment.id;
844         String subId = functionalEvnironment.subId;
845         FunctionalType funType = functionalEvnironment.funType;
846         int seq = functionalEvnironment.seq;
847         Point location = functionalEvnironment.location;
848         List<RecordColumn> inputs = functionalEvnironment.inputTypes;
849         List<RecordColumn> outputs = functionalEvnironment.outputTypes;
850         DataPartFunctional p = env.panel.getNewDataPartFunctional(funType, subId, inputs, outputs)
851         ;

```

```
849         p.setPointBoxLocation(location);
850         env.components.add(new DataPartSeq(seq, p));
851         env.dataPartMap.put(id, p);
852         DataFlowPanel sub = p.getSubDataFlowPanel();
853         dataFlowPanelStack.push(new DataFlowEnvironment(sub, true));
854     }
855     private void endFunctionalSubDataFlow() {
856         dataFlowPanelStack.pop();
857     }
858
859     private void startFunctionalInputs(Attributes attrs) {}
860     private void endFunctionalInputs() {}
861
862     private void startFunctionalOutputs(Attributes attrs) {}
863     private void endFunctionalOutputs() {}
864
865     private void startFunctionalInputColumn(Attributes attrs) {
866         String name = attrs.getValue("id");
867         nextTypeEvent = (type) -> {
868             functionalEvnironment.inputTypes.add(new RecordColumn(name, type));
869         };
870     }
871     private void endFunctionalInputColumn() {
872         nextTypeEvent = null;
873     }
874
875     private void startFunctionalOutputColumn(Attributes attrs) {
876         String name = attrs.getValue("id");
877         nextTypeEvent = (type) -> {
878             functionalEvnironment.outputTypes.add(new RecordColumn(name, type));
879         };
880     }
881     private void endFunctionalOutputColumn() {
882         nextTypeEvent = null;
883     }
884
885     private void startTypeNumber(Attributes attrs) throws InvalidTitleException,
886     InvalidDataPartException {
887         nextTypeEvent.apply(new TypeNumber());
888     }
889     private void endTypeNumber() {
890     }
891
892     private void startTypeBool(Attributes attrs) throws InvalidTitleException,
893     InvalidDataPartException {
894         nextTypeEvent.apply(new TypeBool());
895     }
896
897     private void startTypeString(Attributes attrs) throws InvalidTitleException,
898     InvalidDataPartException {
899         nextTypeEvent.apply(new TypeString());
900     }
901     private void endTypeString() {
```

```
901     }
902
903     private void startTypeTime(Attributes attrs) throws InvalidTitleException,
904         InvalidDataPartException {
905         nextTypeEvent.apply(new TypeTime());
906     }
907     private void endTypeTime() {
908     }
909
910     private void startTypeError(Attributes attrs) throws InvalidTitleException,
911         InvalidDataPartException {
912         nextTypeEvent.apply(new TypeError());
913     }
914     private void endTypeError() {
915     }
916
917     private void startTypeUnknown(Attributes attrs) throws InvalidTitleException,
918         InvalidDataPartException {
919         nextTypeEvent.apply(new TypeUnknown(attrs.getValue("message")));
920     }
921     private void endTypeUnknown() {
922     }
923
924     private void startTypeList(Attributes attrs) {
925         TypeEvent initialTypeEvent = nextTypeEvent;
926         nextTypeEvent = (type) -> {
927             initialTypeEvent.apply(new TypeList(type));
928         };
929     }
930
931     private void endTypeList() {
932         nextTypeEvent = null;
933     }
934
935     private void startTypeAuto(Attributes attrs) {
936         TypeEvent initialTypeEvent = nextTypeEvent;
937         nextTypeEvent = (type) -> {
938             initialTypeEvent.apply(new TypeAuto(type));
939         };
940     }
941     private void endTypeAuto() {
942         nextTypeEvent = null;
943     }
944
945     private void startTypeRecord(Attributes attrs) {
946         typeRecordBuilderStack.push(new TypeRecordBuilder(nextTypeEvent));
947     }
948
949     private void endTypeRecord() throws InvalidTitleException, InvalidDataPartException {
950         TypeRecordBuilder builder = typeRecordBuilderStack.pop();
951         builder.finalEvent.apply(new TypeRecord(builder.columns));
952     }
953
954     private void startRecordColumn(Attributes attrs) {
955         TypeRecordBuilder builder = typeRecordBuilderStack.lastElement();
956         String id = attrs.getValue("id");
957         nextTypeEvent = (type) -> {
```

```
953         builder.columns.add(new RecordColumn(id, type));
954     };
955 }
956 private void endRecordColumn() {
957 }
958
959 private void startConstantString(Attributes attrs)
960     throws InvalidTitleException, InvalidDataPartException {
961     String value = attrs.getValue("value");
962     value = StringEscapeUtils.unescapeXml(value);
963     nextConstantEvent.apply(new ConstantString(value));
964 }
965 private void endConstantString() {
966 }
967
968 private void startConstantRecord(Attributes attrs) {
969     constantRecordBuilderStack.push(new ConstantRecordBuilder(nextConstantEvent));
970 }
971
972 private void endConstantRecord()
973     throws InvalidTitleException, InvalidDataPartException {
974     ConstantRecordBuilder builder = constantRecordBuilderStack.pop();
975     builder.finalEvent.apply(new ConstantRecord(builder.columns));
976 }
977
978 private void startRecordValue(Attributes attrs) {
979     ConstantRecordBuilder builder = constantRecordBuilderStack.lastElement();
980     String id = attrs.getValue("id");
981     nextConstantEvent = (con) -> {
982         builder.columns.add(new RecordValue(id, con));
983     };
984 }
985 private void endRecordValue() {
986 }
987
988 private void startConstantNumber(Attributes attrs)
989     throws InvalidTitleException, InvalidDataPartException {
990     BigDecimal value = new BigDecimal(attrs.getValue("value"));
991     nextConstantEvent.apply(new ConstantNumber(value));
992 }
993 private void endConstantNumber() {
994 }
995
996
997 private void startConstantList(Attributes attrs) {
998     ConstantListBuilder builder = new ConstantListBuilder(nextConstantEvent);
999     constantListBuilderStack.push(builder);
1000    nextTypeEvent = (type) -> {
1001        builder.type = ((TypeList)type).getChildType();
1002    };
1003 }
1004 private void endConstantList() throws InvalidTitleException, InvalidDataPartException {
1005     ConstantListBuilder builder = constantListBuilderStack.pop();
1006     builder.finalEvent.apply(new ConstantList(builder.values, builder.type));
1007     nextTypeEvent = null;
```

```
1008     }
1009
1010    private void startListItem(Attributes attrs) {
1011        ConstantListBuilder builder = constantListBuilderStack.lastElement();
1012        nextConstantEvent = (con) -> {
1013            builder.values.add(con);
1014        };
1015    }
1016    private void endListItem() {
1017    }
1018
1019    private void startConstantBool(Attributes attrs)
1020        throws InvalidTitleException, InvalidDataPartException {
1021        String value = attrs.getValue("value");
1022        nextConstantEvent.apply(new ConstantBool(value.equals("True")));
1023    }
1024    private void endConstantBool() {
1025    }
1026
1027
1028    private void handleElement(String qName, Attributes attrs, boolean isStart)
1029        throws SAXException {
1030        try {
1031            if (qName.equals("ControlFlow")) {
1032                if (isStart) startControlFlow(attrs); else endControlFlow();
1033            } else if (qName.equals("Components")) {
1034                if (isStart) startComponents(attrs); else endComponents();
1035            } else if (qName.equals("Edges")) {
1036                if (isStart) startEdges(attrs); else endEdges();
1037            } else if (qName.equals("Start")) {
1038                if (isStart) startStart(attrs); else endStart();
1039            } else if (qName.equals("Stop")) {
1040                if (isStart) startStop(attrs); else endStop();
1041            } else if (qName.equals("Fail")) {
1042                if (isStart) startFail(attrs); else endFail();
1043            } else if (qName.equals("If")) {
1044                if (isStart) startIf(attrs); else endIf();
1045            } else if (qName.equals("InteractionComponent")) {
1046                if (isStart) startInteractionComponent(attrs); else endInteractionComponent();
1047            } else if (qName.equals("InteractionInputs")) {
1048                if (isStart) startInteractionInputs(attrs); else endInteractionInputs();
1049            } else if (qName.equals("InteractionOutputs")) {
1050                if (isStart) startInteractionOutputs(attrs); else endInteractionOutputs();
1051            } else if (qName.equals("InteractionInput")) {
1052                if (isStart) startInteractionInput(attrs); else endInteractionInput();
1053            } else if (qName.equals("InteractionOutput")) {
1054                if (isStart) startInteractionOutput(attrs); else endInteractionOutput();
1055            } else if (qName.equals("InteractionEvents")) {
1056                if (isStart) startInteractionEvents(attrs); else endInteractionEvents();
1057            } else if (qName.equals("InteractionEvent")) {
1058                if (isStart) startInteractionEvent(attrs); else endInteractionEvent();
1059            } else if (qName.equals("DataFlowComponent")) {
1060                if (isStart) startDataFlowComponent(attrs); else endDataFlowComponent();
1061            } else if (qName.equals("Edge")) {
1062                if (isStart) startEdge(attrs); else endEdge();
```

```
1063     } else if (qName.equals("TypeConnection")) {
1064         if (isStart) startTypeConnection(attrs); else endTypeConnection();
1065     } else if (qName.equals("Connection")) {
1066         if (isStart) startConnection(attrs); else endConnection();
1067     } else if (qName.equals("DataFlow")) {
1068         if (isStart) startDataFlow(attrs); else endDataFlow();
1069     } else if (qName.equals("SinkComponent")) {
1070         if (isStart) startSinkComponent(attrs); else endSinkComponent();
1071     } else if (qName.equals("InputComponent")) {
1072         if (isStart) startInputComponent(attrs); else endInputComponent();
1073     } else if (qName.equals("InputType")) {
1074         if (isStart) startInputType(attrs); else endInputType();
1075     } else if (qName.equals("InputOriginalType")) {
1076         if (isStart) startInputOriginalType(attrs); else endInputOriginalType();
1077     } else if (qName.equals("OutputComponent")) {
1078         if (isStart) startOutputComponent(attrs); else endOutputComponent();
1079     } else if (qName.equals("OutputType")) {
1080         if (isStart) startOutputType(attrs); else endOutputType();
1081     } else if (qName.equals("OutputOriginalType")) {
1082         if (isStart) startOutputOriginalType(attrs); else endOutputOriginalType();
1083     } else if (qName.equals("ComparisonComponent")) {
1084         if (isStart) startComparisonComponent(attrs); else endComparisonComponent();
1085     } else if (qName.equals("ComparisonDataType")) {
1086         if (isStart) startComparisonDataType(attrs); else endComparisonDataType();
1087     } else if (qName.equals("ComparisonOriginalDataType")) {
1088         if (isStart) startComparisonOriginalDataType(attrs); else
endComparisonOriginalDataType();
1089     } else if (qName.equals("ArithmeticComponent")) {
1090         if (isStart) startArithmeticComponent(attrs); else endArithmeticComponent();
1091     } else if (qName.equals("StringOperationComponent")) {
1092         if (isStart) startStringOperationComponent(attrs); else
endStringOperationComponent();
1093     } else if (qName.equals("CastComponent")) {
1094         if (isStart) startCastComponent(attrs); else endCastComponent();
1095     } else if (qName.equals("OriginalCastFrom")) {
1096         if (isStart) startOriginalCastFrom(attrs); else endOriginalCastFrom();
1097     } else if (qName.equals("CastFrom")) {
1098         if (isStart) startCastFrom(attrs); else endCastFrom();
1099     } else if (qName.equals("CastTo")) {
1100         if (isStart) startCastTo(attrs); else endCastTo();
1101     } else if (qName.equals("ListOperationComponent")) {
1102         if (isStart) startListOperationComponent(attrs); else endListOperationComponent();
1103     } else if (qName.equals("ListOperationType")) {
1104         if (isStart) startListOperationType(attrs); else endListOperationType();
1105     } else if (qName.equals("ListOperationOriginalType")) {
1106         if (isStart) startListOperationOriginalType(attrs); else
endListOperationOriginalType();
1107     } else if (qName.equals("ConstantComponent")) {
1108         if (isStart) startConstantComponent(attrs); else endConstantComponent();
1109     } else if (qName.equals("StringFormatComponent")) {
1110         if (isStart) startStringFormatComponent(attrs); else endStringFormatComponent();
1111     } else if (qName.equals("RecordConstructorComponent")) {
1112         if (isStart) startRecordConstructorComponent(attrs); else
endRecordConstructorComponent();
1113     } else if (qName.equals("FunctionalComponent")) {
```

```
1114         if (isStart) startFunctionalComponent(attrs); else endFunctionalComponent();
1115     } else if (qName.equals("FunctionalSubDataFlow")) {
1116         if (isStart) startFunctionalSubDataFlow(attrs); else endFunctionalSubDataFlow();
1117     } else if (qName.equals("FunctionalInputs")) {
1118         if (isStart) startFunctionalInputs(attrs); else endFunctionalInputs();
1119     } else if (qName.equals("FunctionalOutputs")) {
1120         if (isStart) startFunctionalOutputs(attrs); else endFunctionalOutputs();
1121     } else if (qName.equals("FunctionalInputColumn")) {
1122         if (isStart) startFunctionalInputColumn(attrs); else endFunctionalInputColumn();
1123     } else if (qName.equals("FunctionalOutputColumn")) {
1124         if (isStart) startFunctionalOutputColumn(attrs); else endFunctionalOutputColumn();
1125     } else if (qName.equals("TypeNumber")) {
1126         if (isStart) startTypeNumber(attrs); else endTypeNumber();
1127     } else if (qName.equals("TypeBool")) {
1128         if (isStart) startTypeBool(attrs); else endTypeBool();
1129     } else if (qName.equals("TypeString")) {
1130         if (isStart) startTypeString(attrs); else endTypeString();
1131     } else if (qName.equals("TypeTime")) {
1132         if (isStart) startTypeTime(attrs); else endTypeTime();
1133     } else if (qName.equals("TypeError")) {
1134         if (isStart) startTypeError(attrs); else endTypeError();
1135     } else if (qName.equals("TypeUnknown")) {
1136         if (isStart) startTypeUnknown(attrs); else endTypeUnknown();
1137     } else if (qName.equals("TypeList")) {
1138         if (isStart) startTypeList(attrs); else endTypeList();
1139     } else if (qName.equals("TypeAuto")) {
1140         if (isStart) startTypeAuto(attrs); else endTypeAuto();
1141     } else if (qName.equals("TypeRecord")) {
1142         if (isStart) startTypeRecord(attrs); else endTypeRecord();
1143     } else if (qName.equals("RecordColumn")) {
1144         if (isStart) startRecordColumn(attrs); else endRecordColumn();
1145     } else if (qName.equals("ConstantString")) {
1146         if (isStart) startConstantString(attrs); else endConstantString();
1147     } else if (qName.equals("ConstantRecord")) {
1148         if (isStart) startConstantRecord(attrs); else endConstantRecord();
1149     } else if (qName.equals("RecordValue")) {
1150         if (isStart) startRecordValue(attrs); else endRecordValue();
1151     } else if (qName.equals("ConstantNumber")) {
1152         if (isStart) startConstantNumber(attrs); else endConstantNumber();
1153     } else if (qName.equals("ConstantList")) {
1154         if (isStart) startConstantList(attrs); else endConstantList();
1155     } else if (qName.equals("ListItem")) {
1156         if (isStart) startListItem(attrs); else endListItem();
1157     } else if (qName.equals("ConstantBool")) {
1158         if (isStart) startConstantBool(attrs); else endConstantBool();
1159     }
1160     } catch (InvalidTitleException | InvalidDataPartException e) {
1161         throw new SAXException(e);
1162     }
1163 }
1164
1165 @Override
1166 public void startElement(String uri, String localName, String qName, Attributes attributes)
1167 throws SAXException {
    handleElement(qName, attributes, true);
```

```

1168     }
1169
1170     @Override
1171     public void endElement(String uri, String localName, String qName) throws SAXException {
1172         handleElement(qName, null, false);
1173     }
1174
1175     public static ControlFlowPanel open(File file, MainWindow mainWin) throws ASTLoaderException {
1176         SAXParserFactory factory = SAXParserFactory.newInstance();
1177         factory.setValidating(true);
1178         try {
1179             SAXParser parser = factory.newSAXParser();
1180             ASTLoader ast = new ASTLoader(mainWin);
1181             parser.parse(file, ast);
1182             return ast.rootControlFlow;
1183         } catch (ParserConfigurationException | SAXException | IOException e) {
1184             throw new ASTLoaderException(e.getMessage());
1185         }
1186     }
1187
1188     @SuppressWarnings("serial")
1189     public static class ASTLoaderException extends Exception {
1190         public ASTLoaderException(String msg) {
1191             super(msg);
1192         }
1193     }
1194 }
```

LISTING E.102: gui/ASTLoader.java

```

1 package gui;
2
3 import java.awt.BorderLayout;
4 import java.awt.Color;
5 import java.awt.Dimension;
6 import java.awt.FlowLayout;
7 import java.awt.Graphics;
8 import java.awt.Graphics2D;
9 import java.awt.GridBagLayout;
10 import java.awt.Point;
11 import java.awt.Rectangle;
12 import java.awt.RenderingHints;
13 import java.awt.event.ActionEvent;
14 import java.awt.event.ActionListener;
15 import java.awt.event.MouseEvent;
16 import java.awt.event.MouseListener;
17 import java.awt.event.MouseMotionListener;
18 import java.util.ArrayList;
19 import java.util.Arrays;
20 import java.util.Collection;
21 import java.util.LinkedList;
22 import java.util.List;
23 import java.util.Map;
24 import java.util.Set;
25 import java.util.function.Predicate;
```

```
26
27 import gui.controlflow.ControlArrow;
28 import gui.controlflow.ControlPart;
29 import gui.controlflow.ControlPartTypeConnection;
30 import gui.controlflow.GraphUtil;
31 import gui.controlflow.GraphUtil.Dominator;
32 import gui.dataflow.DataPartParameter;
33 import gui.dataflow.DataPartResult;
34 import lang.type.RecordColumn;
35
36 import javax.swing.BoxLayout;
37 import javax.swing.Icon;
38 import javax.swing.JLabel;
39 import javax.swing.JMenuItem;
40 import javax.swing.JOptionPane;
41 import javax.swing.JPanel;
42 import javax.swing.JPopupMenu;
43 import javax.swing.JSeparator;
44 import javax.swing.SwingConstants;
45
46 import util.ColorTheme;
47 import util.IconUtil;
48 import util.UtilButton;
49 import util.UtilFont;
50 import util.UtilScrollPane;
51
52 @SuppressWarnings("serial")
53 public class ControlFlowPanel extends TabbedPane implements
54     MouseMotionListener, MouseListener {
55     private List<ControlPart> controlParts = new LinkedList<>();
56     private ArrayList<ControlArrow> controlArrows = new ArrayList<>();
57     private int focusCircle = -1;
58     private int selectedCircle = -1;
59     private ControlPart focusCircleControlPart = null;
60     private ControlPart selectedCircleControlPart = null;
61     private ControlPart focusControlPart = null;
62     private ArrayList<ControlPart> selectedControlParts = new ArrayList<>();
63     private Point dragStart = null;
64     private ControlArrow dragArrow;
65     private ControlArrow focusArrow;
66     private ControlArrow selectedArrow;
67     private Rectangle dragRectangle;
68     private JPopupMenu popupMenu;
69
70     private JPopupMenu selectionPopupMenu = new JPopupMenu();
71     private UtilScrollPane selectionPopupScrollPane;
72     private ControlPart selectionStateControlPart;
73     private String selectionStateAgumentName;
74     private ControlArrow selectionStateControlArrow;
75     private ControlPart selectionPopupMenuControlPart;
76     // When != null we are in selection mode.
77     private List<Dominator> selectionStateVisibleControlParts;
78
79     private final static Icon controlFlowIcon = IconUtil.createImageIcon("icons/letter-c-16.png");
80     private boolean mouseIsPressed = false;
```

```
81     private ControlPart startControlPart;
82
83     private int nextSequenceNumber = 0;
84
85     private String programPlatform;
86     private String programUserGroup;
87
88     public ControlFlowPanel(MainWindow mainWindow, String title, String platform, String group,
89                             TabbedPanel parent)
90         throws InvalidTitleException {
91         super(mainWindow, title, parent);
92
93         setBackground(Color.WHITE);
94
95         selectionPopupMenu.setLayout(new BorderLayout());
96
97         popupMenu = new JPopupMenu();
98         JMenuItem buttonDelete = new JMenuItem("Delete");
99         popupMenu.add(buttonDelete);
100        buttonDelete.addActionListener(new ActionListener() {
101            @Override
102            public void actionPerformed(ActionEvent e) {
103                deleteEvent();
104            }
105        });
106
107        setPlatform(platform);
108        setUserGroup(group);
109
110        startControlPart = ControlPart.getStartControlPart(this, mainWindow.getGraphics());
111        startControlPart.setSequenceNumber(nextSequenceNumber++);
112        controlParts.add(startControlPart);
113        startControlPart.setLocation(new Point(100, 40));
114
115        addMouseListener(this);
116        addMouseMotionListener(this);
117    }
118
119    public String getPlatform() {
120        return programPlatform;
121    }
122
123    public String getUserGroup() {
124        return programUserGroup;
125    }
126
127    public void setPlatform(String platform) throws InvalidTitleException {
128        MainWindow.validatePlatform(platform);
129        programPlatform = platform;
130    }
131
132    public void setUserGroup(String group) throws InvalidTitleException {
133        MainWindow.validateUserGroup(group);
134        programUserGroup = group;
135    }
```

```
135
136     void setNonStartControlParts(List<ControlPart> ps) {
137         controlParts = new ArrayList<>();
138         controlParts.add(startControlPart);
139         controlParts.addAll(ps);
140     }
141
142     public ControlPart getStartControlPart() {
143         return startControlPart;
144     }
145
146     List<ControlPart> getControlParts() {
147         return controlParts;
148     }
149
150     List<ControlArrow> getControlArrows() {
151         return controlArrows;
152     }
153
154     private boolean deleteArrow(ControlArrow a, boolean confirmDelete) {
155         List<ControlArrow> affected = GraphUtil.getDeleteAffectedArrows(a, controlParts,
156         controlArrows);
157
158         int confirm = 0;
159         if (confirmDelete && affected.size() > 0) {
160             confirm = JOptionPane.showOptionDialog(mainWindow,
161                 "Delete will affect input for indirectly connected " +
162                 "control flow component(s).\\nDo you want to delete?",
163                 "Delete confirmation", JOptionPane.YES_NO_OPTION,
164                 JOptionPane.QUESTION_MESSAGE, null, null, null);
165         }
166         if (confirm == 0) {
167             controlArrows.remove(a);
168             for (ControlArrow aff : affected) {
169                 ControlPart startPart = aff.getStartControlPart();
170                 ControlPart endPart = aff.getEndControlPart();
171                 Set<ControlPart> doms = GraphUtil.getDominatorControlPartSet(startPart, endPart,
172                 controlParts, controlArrows);
173                 for (ControlPartTypeConnection c : aff.getArgumentConnections().values()) {
174                     if (!doms.contains(c.getOutputPart())) {
175                         aff.removeArgumentConnection(c.getInputName());
176                     }
177                 }
178             }
179         }
180     }
181
182     private boolean deleteControlParts(List<ControlPart> parts) {
183         if (parts.contains(startControlPart)) {
184             JOptionPane.showMessageDialog(mainWindow, "Cannot delete 'Start'.",
185                 "Unable to delete", JOptionPane.ERROR_MESSAGE);
186         }
187     }

```

```
188     ArrayList<ControlArrow> connectedArrows = new ArrayList<>();
189     for (int i = 0; i < controlArrows.size(); i++) {
190         ControlArrow a = controlArrows.get(i);
191         for (ControlPart cp : parts) {
192             if (a.getStartControlPart() == cp || a.getEndControlPart() == cp) {
193                 connectedArrows.add(a);
194             }
195         }
196     }
197 }
198
199     int confirm = 0;
200     for (ControlArrow a : connectedArrows) {
201         List<ControlArrow> affected = GraphUtil.getDeleteAffectedArrows(a, controlParts,
202         controlArrows);
203         if (affected.size() > 0) {
204             confirm = JOptionPane.showOptionDialog(mainWindow,
205                 "Delete will affect input for indirectly connected " +
206                 "control flow component(s).\nDo you want to delete?",
207                 "Delete confirmation", JOptionPane.YES_NO_OPTION,
208                 JOptionPane.QUESTION_MESSAGE, null, null, null);
209             break;
210         }
211     }
212     if (confirm != 0)
213         return false;
214
215     for (ControlArrow a : connectedArrows) {
216         deleteArrow(a, false);
217     }
218     for (ControlPart p : parts) {
219         TabbedPane panel = p.getTabbedPane();
220         if (panel != null)
221             mainWindow.removeExplorerItem(panel);
222         p.markDeleted();
223         controlParts.remove(p);
224     }
225     return true;
226 }
227
228 public void deleteEvent() {
229     if (selectedArrow != null && deleteArrow(selectedArrow, true)) {
230         selectedArrow = null;
231     }
232     if (!selectedControlParts.isEmpty() && deleteControlParts(selectedControlParts)) {
233         selectedControlParts.clear();
234     }
235     repaint();
236 }
237
238 private void maybeShowPopup(MouseEvent e) {
239     if (e.isPopupTrigger() && (selectedArrow != null || !selectedControlParts.isEmpty())) {
240         Point p = transformPoint(getTransform(), e.getPoint());
241         popupMenu.show(e.getComponent(), p.x, p.y);
```

```
242     }
243 }
244
245 public void addControlArrow(ControlArrow a) {
246     controlArrows.add(a);
247     repaint();
248 }
249
250 public void addControlPart(ControlPart part) {
251     addControlPart(part, true);
252 }
253
254 public void addControlPart(ControlPart part, boolean setLocation) {
255     if (setLocation) {
256         Rectangle screen = scrollPane.getViewport().getViewRect();
257         int w = inverseZoom(screen.width/2.0);
258         int h = inverseZoom(screen.height/2.0);
259         part.setLocation(new Point(screen.x + w, screen.y + h));
260     }
261     controlParts.add(0, part);
262     part.setSequenceNumber(nextSequenceNumber++);
263     repaint();
264 }
265
266 private void validateAddDataflowControlPart(ControlPart dataFlow)
267     throws InvalidTitleException {
268     for (ControlPart p : controlParts) {
269         if (p.isDataFlow() && p.getTitle().equals(dataFlow.getTitle()))
270             throw new InvalidTitleException("Dataflow title already exists "+p.getTitle()+"");
271     }
272 }
273
274 public ControlPart addNewDataflowControlPart(String title) throws InvalidTitleException {
275     ControlPart p = ControlPart.getDataflowControlPart(mainWindow, title, this, mainWindow.
276     getGraphics());
277     validateAddDataflowControlPart(p);
278     TabbedPane panel = p.getTabbedPane();
279     mainWindow.addExplorerItem(this, panel);
280     addControlPart(p);
281     return p;
282 }
283
284 public ControlPart addNewInteractionControlPart(String title, String platform, String[] events
285     ,
286     List<RecordColumn> inputs, List<RecordColumn> outputs) {
287     ControlPart p = ControlPart.getInteractionControlPart(title, platform, events,
288         inputs, outputs, this, mainWindow.getGraphics());
289     addControlPart(p);
290     return p;
291 }
292
293 public ControlPart addNewIfRuntimeInteractionControlPart() {
294     ControlPart p = ControlPart.getIfRuntimeInteractionControlPart(this, mainWindow.
295     getGraphics());
```

```
293         addControlPart(p);
294         return p;
295     }
296
297     public ControlPart addNewStopControlPart() {
298         ControlPart p = ControlPart.getStopControlPart(this, mainWindow.getGraphics());
299         addControlPart(p);
300         return p;
301     }
302
303     public ControlPart addNewFailControlPart() {
304         ControlPart p = ControlPart.getFailControlPart(this, mainWindow.getGraphics());
305         addControlPart(p);
306         return p;
307     }
308
309     @Override
310     public void paintComponent(Graphics g) {
311         Graphics2D g2 = (Graphics2D) g;
312         super.paintComponent(g);
313         g2.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING, RenderingHints.
314             VALUE_TEXT_ANTIALIAS_ON);
315         g2.transform(getTransform());
316
317         if (selectionStateVisibleControlParts == null)
318             normalPaint(g2);
319         else
320             selectionPaint(g2);
321
322     }
323
324     private void normalPaint(Graphics2D g2){
325         for (int i = controlArrows.size() - 1; i >= 0; i--)
326             controlArrows.get(i).paint(g2);
327         if (dragArrow != null)
328             dragArrow.paint(g2);
329         for (int i = controlParts.size() - 1; i >= 0; i--) {
330             ControlPart cp = controlParts.get(i);
331             cp.paint(g2, true);
332         }
333         if (dragRectangle != null) {
334             Rectangle r = getDrawRectangle(dragRectangle);
335             Color c = g2.getColor();
336             g2.setColor(ColorTheme.TRANS_LIGHT_GRAY);
337             g2.fill(r);
338             g2.setColor(ColorTheme.NONTRANS_LIGHT_GRAY);
339             g2.draw(r);
340             g2.setColor(c);
341         }
342     }
343
344     private void selectionPaint(Graphics2D g2) {
345         for (int i = controlArrows.size() - 1; i >= 0; i--)
346             controlArrows.get(i).paint(g2);
```

```
347     if (dragArrow != null)
348         dragArrow.paint(g2);
349     for (int i = controlParts.size() - 1; i >= 0; i--) {
350         ControlPart cp = controlParts.get(i);
351         if (!isVisibleSelectionControlPart(cp))
352             cp.paint(g2, false);
353     }
354     g2.setColor(new Color(0xD4, 0xCF, 0xD4, 127));
355     Rectangle viewRect = transformRectangle(getInverseTransform(), scrollPane.setViewport().
356     getViewRect());
357     g2.fill(viewRect);
358     for (int i = selectionStateVisibleControlParts.size() - 1; i >= 0; i--) {
359         ControlPart cp = selectionStateVisibleControlParts.get(i).getControlPart();
360         cp.paint(g2, false);
361     }
362     g2.setColor(Color.black);
363 }
364
365 private boolean isThereExistingArrow(ControlPart controlPart, int circleIndex) {
366     for (ControlArrow a : controlArrows) {
367         ControlPart p = a.getStartControlPart();
368         if (p != controlPart)
369             continue;
370
371         int index = a.getStartCircleIndex();
372         int first, second;
373         if (circleIndex % 2 == 0) {
374             first = circleIndex;
375         } else {
376             first = circleIndex - 1;
377         }
378         second = first + 1;
379
380         if (first == index || second == index) {
381             return true;
382         }
383     }
384     return false;
385 }
386
387 private void selectedComponentReset(boolean clearSelectedControlParts) {
388     selectedCircle = -1;
389     selectedCircleControlPart = null;
390     dragRectangle = null;
391
392     if (selectedControlParts.size() > 0) {
393         for (ControlPart p : selectedControlParts)
394             p.setDrawSelectedBorder(false);
395         if (clearSelectedControlParts)
396             selectedControlParts.clear();
397     }
398     if (selectedArrow != null) {
399         selectedArrow.setSelected(false);
400         selectedArrow = null;
401     }
402 }
```

```
401    }
402
403    private void focusComponentReset() {
404        focusCircle = -1;
405        if (focusCircleControlPart != null) {
406            focusCircleControlPart.setFocusCircle(-1);
407            focusCircleControlPart = null;
408        }
409        if (focusControlPart != null) {
410            focusControlPart.setDrawFocusBorder(false);
411            focusControlPart = null;
412        }
413        if (focusArrow != null) {
414            focusArrow.setHovered(false);
415            focusArrow = null;
416        }
417    }
418
419    private void normalPress(MouseEvent e) {
420        Point point = e.getPoint();
421        boolean clearControlParts = true;
422        for (ControlPart p : selectedControlParts) {
423            if (p.isOverControlPart(point)) {
424                clearControlParts = false;
425                break;
426            }
427        }
428
429        selectedComponentReset(clearControlParts);
430
431        dragStart = point;
432
433        ControlPart clickControlPart = null;
434        for (ControlPart cp : controlParts) {
435            selectedCircle = cp.getCircleIndex(point);
436            if (selectedCircle >= 0) {
437                if (!isThereExistingArrow(cp, selectedCircle)) {
438                    selectedCircleControlPart = cp;
439                    selectedControlParts.clear();
440                    break;
441                }
442            }
443            if (cp.isOverControlPart(point)) {
444                clickControlPart = cp;
445                if (clearControlParts)
446                    selectedControlParts.add(cp);
447                break;
448            }
449        }
450        if (selectedControlParts.isEmpty() && selectedCircleControlPart == null) {
451            for (ControlArrow a : controlArrows) {
452                if (a.intersects(point)) {
453                    selectedArrow = a;
454                    break;
455                }
456            }
457        }
458    }
459
```

```
456         }
457     }
458     if (selectedControlParts.size() > 0) {
459         for (int i = selectedControlParts.size()-1; i >= 0; i--) {
460             ControlPart p = selectedControlParts.get(i);
461             p.setDrawSelectedBorder(true);
462         }
463         if (clickControlPart != null) {
464             controlParts.remove(clickControlPart);
465             controlParts.add(0, clickControlPart);
466         }
467     }
468     if (selectedCircleControlPart != null) {
469         if (!isThereExistingArrow(selectedCircleControlPart, selectedCircle)) {
470             dragArrow = new ControlArrow(selectedCircleControlPart,
471                 selectedCircle, point);
472         } else {
473             selectedCircle = -1;
474             selectedCircleControlPart = null;
475         }
476     }
477     if (selectedArrow != null) {
478         selectedArrow.setSelected(true);
479         controlArrows.remove(selectedArrow);
480         controlArrows.add(0, selectedArrow);
481     }
482
483     maybeShowPopup(e);
484     if (!e.isPopupTrigger() && e.getClickCount() == 2) {
485         if (clickControlPart != null) {
486             TabbedPane panel = clickControlPart.getTabbedPane();
487             for (ControlPart part : selectedControlParts) {
488                 part.setDrawSelectedBorder(false);
489             }
490             selectedControlParts.clear();
491             selectedControlParts.add(clickControlPart);
492             clickControlPart.setDrawSelectedBorder(true);
493             if (panel != null)
494                 mainWindow.openExplorerItem(panel);
495         } else if (selectedArrow != null) {
496             selectedArrow.openTypeMapper(mainWindow, this);
497         }
498     }
499
500     if (selectedCircleControlPart == null &&
501         selectedControlParts.isEmpty() && selectedArrow == null) {
502         dragRectangle = new Rectangle(point.x, point.y, 0, 0);
503     }
504 }
505 private void selectionPress(MouseEvent e) {
506     Point point = e.getPoint();
507     boolean stopSelectionMode = true;
508
509     for (ControlPart cp : controlParts) {
510         if (cp.isOverControlPart(point)) {
```

```
511             stopSelectionMode = false;
512         break;
513     }
514 }
515
516 if (stopSelectionMode) {
517     selectionStateControlArrow.openTypeMapper(mainWindow, this);
518 }
519 }
520
521 private void showSelectionPopup(Point point, ControlPart part) {
522     List<RecordColumn> outputs = null;
523
524     for (Dominator dom : selectionStateVisibleControlParts) {
525         if (dom.getControlPart() != part)
526             continue;
527         if (dom.isErrorPathDominator())
528             outputs = new ArrayList<>(Arrays.asList(ControlPart.errorRecordColumn));
529         else
530             outputs = part.getOutputs(false);
531     }
532
533     if (outputs.size() == 0)
534         return;
535
536     JPanel panel = new JPanel();
537     panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
538     for (int index = 0; index < outputs.size(); index++) {
539         RecordColumn row = outputs.get(index);
540         JPanel inside = new JPanel();
541         inside.setLayout(new BoxLayout(inside, BoxLayout.Y_AXIS));
542         JPanel labelPanel = new JPanel(new GridBagLayout());
543         final String outputId = row.getId();
544         JLabel label = new JLabel(outputId);
545         label.setFont(UtilFont.boldTextFont);
546         labelPanel.add(label);
547         inside.add(labelPanel);
548         ShowTypePanel showPanel = new ShowTypePanel(row.getType());
549         JPanel showContainer = new JPanel(new GridBagLayout());
550         showContainer.add(showPanel);
551         inside.add(showContainer);
552         JPanel selectPanel = new JPanel(new BorderLayout());
553         selectPanel.add(inside);
554         UtilButton select = new UtilButton("Select");
555         select.addActionListener(new ActionListener() {
556             @Override
557             public void actionPerformed(ActionEvent e) {
558                 ControlPartTypeConnection conn = new ControlPartTypeConnection(part, outputId,
559                             selectionStateControlPart, selectionStateArgumentName, true);
560                 selectionStateControlArrow.putArgumentConnection(conn);
561                 selectionPopupMenu.remove(selectionPopupMenuScrollPane);
562                 selectionPopupMenu.setVisible(false);
563                 selectionPopupMenuScrollPane = null;
564
565                 selectionStateControlArrow.openTypeMapper(mainWindow, ControlFlowPanel.this);
566             }
567         });
568     }
569 }
```

```
566             //new TypeMapper(mainWindow, ControlFlowPanel.this, selectionStateControlPart)
567         ;
568     }
569     JPanel southPanel = new JPanel(new GridBagLayout());
570     southPanel.add(select);
571     selectPanel.add(southPanel, BorderLayout.SOUTH);
572     JPanel bounder = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));
573     bounder.add(selectPanel);
574     panel.add(bounder);
575     if (index < outputs.size()-1)
576         panel.add(new JSeparator(SwingConstants.HORIZONTAL));
577 }
578
579 if (selectionPopupMenu != null)
580     selectionPopupMenu.remove(selectionPopupScrollPane);
581
582 selectionPopupScrollPane = new UtilScrollPane(panel);
583 Dimension prefSize = selectionPopupScrollPane.getPreferredSize();
584 boolean resetSize = false;
585 if (prefSize.width > 400) {
586     prefSize.width = 400;
587     prefSize.height += 30;
588     resetSize = true;
589 }
590 if (prefSize.height > 300) {
591     prefSize.height = 300;
592     prefSize.width += 30;
593     resetSize = true;
594 }
595 if (resetSize)
596     selectionPopupScrollPane.setPreferredSize(prefSize);
597 selectionPopupMenu.add(selectionPopupScrollPane);
598
599 selectionPopupMenu.revalidate();
600 point = transformPoint(getTransform(), point);
601 selectionPopupMenu.show(this, point.x-30, point.y-30);
602 }
603
604 @Override
605 public void mousePressed(MouseEvent e1) {
606     if (isOverSideOval(e1.getPoint())) {
607         repaint();
608         return;
609     }
610     MouseEvent e = inverseTransformMouseEvent(e1);
611     mouseIsPressed = true;
612     if (selectionStateVisibleControlParts == null)
613         normalPress(e);
614     else
615         selectionPress(e);
616     repaint();
617 }
618
619 private boolean validControlPartArguments() {
```

```
620     List<ControlArrow> arrows = GraphUtil.getAllBrokenArgumentArrows(controlParts,
621     controlArrows);
622     if (arrows.size() == 0)
623         return true;
624
625     int confirm = JOptionPane.showOptionDialog(mainWindow,
626         "New path invalidates existing input for component(s).\n" +
627         "Do you want to keep the new path?",
628         "New path confirmation", JOptionPane.YES_NO_OPTION,
629         JOptionPane.QUESTION_MESSAGE, null, null);
630     if (confirm != 0)
631         return false;
632
633     for (ControlArrow a : arrows) {
634         ControlPart startPart = a.getStartControlPart();
635         ControlPart endPart = a.getEndControlPart();
636         Set<ControlPart> doms = GraphUtil.getDominatorControlPartSet(startPart, endPart,
637         controlParts, controlArrows);
638         for (ControlPartTypeConnection c : a.getArgumentConnections().values()) {
639             if (!doms.contains(c.getOutputPart())) {
640                 a.removeArgumentConnection(c.getInputName());
641             }
642         }
643     }
644
645     return true;
646 }
647
648 private void normalRelease(MouseEvent e) {
649     Point point = e.getPoint();
650     dragArrow = null;
651     mouseIsPressed = false;
652
653     if (selectedCircleControlPart != null) {
654         for (ControlPart cp : controlParts) {
655             if (cp.isOverControlPart(point)) {
656                 ControlArrow arrow = new ControlArrow(
657                     selectedCircleControlPart, selectedCircle, cp);
658                 controlArrows.add(0, arrow);
659                 if (validControlPartArguments())
660                     arrow.openTypeMapper(mainWindow, this);
661                 else {
662                     controlArrows.remove(0);
663                 }
664             }
665         }
666
667     if (dragRectangle != null) {
668         Rectangle r = getDrawRectangle(dragRectangle);
669         for (ControlPart cp : controlParts) {
670             if (cp.isInsideOrIntersects(r)) {
671                 selectedControlParts.add(cp);
672             }
673         }
674     }
675 }
```

```
673         }
674         for (ControlPart cp : selectedControlParts) {
675             cp.setDrawSelectedBorder(true);
676             controlParts.remove(cp);
677             controlParts.add(0, cp);
678         }
679         dragRectangle = null;
680     }
681
682     maybeShowPopup(e);
683 }
684 private void selectionRelease(MouseEvent e) {
685 }
686
687 @Override
688 public void mouseReleased(MouseEvent e1) {
689     //if (isOverSideOval(e1.getPoint())) {
690     //    repaint();
691     //    return;
692     //}
693     MouseEvent e = inverseTransformMouseEvent(e1);
694     if (selectionStateVisibleControlParts == null)
695         normalRelease(e);
696     else
697         selectionRelease(e);
698     resetPreferredSize();
699     repaint();
700 }
701
702 @Override
703 public void mouseDragged(MouseEvent e1) {
704     if (isOverSideOval(e1.getPoint())) {
705         repaint();
706         return;
707     }
708     applyMouseDrag(e1.getPoint());
709     repaint();
710 }
711
712 @Override
713 public void applyMouseDrag(Point p1) {
714     if (!mouseIsPressed)
715         return;
716     Point point = inverseTransformPoint(p1);
717     if (selectionStateVisibleControlParts == null)
718         normalDrag(point);
719     else
720         selectionDrag(point);
721 }
722
723 private void normalDrag(Point point) {
724     setFocusComponent(point);
725     if (!selectedControlParts.isEmpty()) {
726         Point delta = new Point(point.x - dragStart.x, point.y - dragStart.y);
727         for (ControlPart p : selectedControlParts)
```

```
728             p.moveLocation(delta);
729         }
730         if (dragRectangle != null) {
731             int x = dragRectangle.x;
732             int y = dragRectangle.y;
733             int w = dragRectangle.width + (point.x-dragStart.x);
734             int h = dragRectangle.height + (point.y-dragStart.y);
735             dragRectangle.setBounds(x, y, w, h);
736         }
737         dragStart = point;
738     }
739     if (dragArrow != null) {
740         dragArrow = null;
741         for (ControlPart cp : controlParts) {
742             if (cp.isOverControlPart(point)) {
743                 dragArrow = new ControlArrow(selectedCircleControlPart,
744                     selectedCircle, cp);
745                 break;
746             }
747         }
748         if (dragArrow == null) {
749             dragArrow = new ControlArrow(selectedCircleControlPart,
750                 selectedCircle, point);
751         }
752     }
753 }
754 private void selectionDrag(Point p) {
755     selectionMove(p);
756 }
757
758 @Override
759 public void mouseMoved(MouseEvent e1) {
760     if (isOverSideOval(e1.getPoint())) {
761         repaint();
762         return;
763     }
764     MouseEvent e = inverseTransformMouseEvent(e1);
765     if (selectionStateVisibleControlParts == null)
766         normalMove(e);
767     else
768         selectionMove(e.getPoint());
769     repaint();
770 }
771
772 private void normalMove(MouseEvent e) {
773     setFocusComponent(e.getPoint());
774 }
775 private void selectionMove(Point point) {
776     focusComponentReset();
777     for (Dominator cp : selectionStateVisibleControlParts) {
778         if (cp.getControlPart().isOverControlPart(point))
779             focusControlPart = cp.getControlPart();
780             break;
781     }
782 }
```

```
783     if (focusControlPart != null) {
784         if (selectionPopupScrollPane == null ||
785             selectionPopupMenuControlPart != focusControlPart) {
786             showSelectionPopup(point, focusControlPart);
787             selectionPopupMenuControlPart = focusControlPart;
788         }
789         focusControlPart.setDrawFocusBorder(true);
790     } else if (selectionPopupScrollPane != null) {
791         selectionPopupMenu.setVisible(false);
792         selectionPopupMenu.remove(selectionPopupScrollPane);
793         selectionPopupScrollPane = null;
794     }
795 }
796
797 private void setFocusComponent(Point point) {
798     focusComponentReset();
799
800     for (ControlPart cp : controlParts) {
801         focusCircle = cp.getCircleIndex(point);
802         if (focusCircle >= 0) {
803             if (!isThereExistingArrow(cp, focusCircle)) {
804                 focusCircleControlPart = cp;
805                 break;
806             }
807         }
808         if (cp.isOverControlPart(point)) {
809             focusControlPart = cp;
810             break;
811         }
812     }
813
814     if (focusCircleControlPart == null && focusControlPart == null) {
815         for (ControlArrow a : controlArrows) {
816             if (a.intersects(point)) {
817                 focusArrow = a;
818                 break;
819             }
820         }
821     }
822
823     if (focusCircleControlPart != null)
824         focusCircleControlPart.setFocusCircle(focusCircle);
825     if (focusControlPart != null) {
826         focusControlPart.setDrawFocusBorder(true);
827     }
828     if (focusArrow != null)
829         focusArrow.setHovered(true);
830 }
831
832 public void enterSelectionState(ControlArrow arrow, String argumentName) {
833     selectionStateArgumentName = argumentName;
834     selectionStateControlPart = arrow.getEndControlPart();
835     selectionStateControlArrow = arrow;
836     selectionStateVisibleControlParts = GraphUtil.getDominators(arrow.getStartControlPart(),
837         arrow.getEndControlPart(), controlParts, controlArrows);
```

```
838     focusComponentReset();
839     selectedComponentReset(true);
840     repaint();
841 }
842
843 public void leaveSelectionState() {
844     if (selectionStateVisibleControlParts != null) {
845         selectionStateVisibleControlParts = null;
846         selectionStateArgumentName = null;
847         selectionStateControlPart = null;
848         selectionStateControlArrow = null;
849         focusComponentReset();
850         selectedComponentReset(true);
851         repaint();
852     }
853 }
854
855 public boolean isVisibleSelectionControlPart(ControlPart p) {
856     return selectionStateVisibleControlParts.contains(p);
857 }
858
859 @Override
860 public Icon getIcon() {
861     return controlFlowIcon;
862 }
863
864 private ControlPart getTabbedPaneControlPart(TabbedPane p) {
865     for (ControlPart c : controlParts) {
866         if (c.getTabbedPane() == p)
867             return c;
868     }
869     return null;
870 }
871
872 @Override
873 public Predicate<DataPartParameter> getChildParamaterDeleteEvent() {
874     return (param) -> {
875         ControlPart controlPart = getTabbedPaneControlPart(param.getDataFlowPanel());
876         if (controlPart == null)
877             return true;
878
879         List<ControlArrow> arrows = GraphUtil.getInputArrowMap(controlParts, controlArrows).
880         get(controlPart);
881         boolean acceptDelete = false;
882         for (ControlArrow a : arrows) {
883             if (a.getArgumentConnections().containsKey(param.getName())) {
884                 if (!acceptDelete) {
885                     int confirm = JOptionPane.showOptionDialog(mainWindow,
886                         "Delete will affect controlflow "+getTitle()+".\nDo you want to
887                         delete?", "Delete confirmation", JOptionPane.YES_NO_OPTION,
888                         JOptionPane.QUESTION_MESSAGE, null, null);
889                     if (confirm != 0)
890                         return false;
891                     acceptDelete = true;
892                 }
893             }
894         }
895     };
896 }
897
898 @Override
899 public void componentResized(ComponentEvent e) {
900     if (e.getSource() == mainWindow)
901         mainPanel.revalidate();
902 }
```

```
891         }
892         a.removeArgumentConnection(param.getName());
893     }
894 }
895 return true;
896 };
897 }
898
899 @Override
900 public Predicate<DataPartResult> getChildResultDeleteEvent() {
901     return (result) -> {
902         ControlPart controlPart = getTabbedPanelControlPart(result.getDataFlowPanel());
903         if (controlPart == null)
904             return true;
905
906         boolean acceptDelete = false;
907         for (ControlArrow a : controlArrows) {
908             Collection<ControlPartTypeConnection> conns = a.getArgumentConnections().values();
909             for (ControlPartTypeConnection conn : conns) {
910                 if (conn.getOutputPart() == controlPart && conn.getOutputName().equals(result.
911                     getName())) {
912                     if (!acceptDelete) {
913                         int confirm = JOptionPane.showOptionDialog(mainWindow,
914                             "Delete will affect controlflow '"+getTitle()+"'.\nDo you want
915                             to delete?",
916                             "Delete confirmation", JOptionPane.YES_NO_OPTION,
917                             JOptionPane.QUESTION_MESSAGE, null, null, null);
918                     if (confirm != 0)
919                         return false;
920                     acceptDelete = true;
921                 }
922             }
923         }
924     return true;
925 };
926 }
927
928 @Override
929 public void mouseClicked(MouseEvent e) {}
930 @Override
931 public void mouseEntered(MouseEvent e) {}
932 @Override
933 public void mouseExited(MouseEvent e) {}
934
935 @Override
936 public Dimension getMinimumDimension() {
937     Dimension min = new Dimension(0,0);
938     for (ControlPart p : controlParts) {
939         Point loc = p.getLocation();
940         Dimension size = p.getSize();
941         min.width = Math.max(loc.x+size.width, min.width);
942         min.height = Math.max(loc.y+size.height, min.height);
943     }

```

```
944         min.width += 60;
945         min.height += 60;
946         return min;
947     }
948
949     @Override
950     public void validateDuplicateTitle(String title)
951         throws InvalidTitleException {
952     }
953
954     @Override
955     public boolean hasDataFlowWithTitle(String title) {
956         for (ControlPart p : controlParts) {
957             if (p.isDataFlow() && p.getTitle().equals(title))
958                 return true;
959         }
960         return false;
961     }
962
963     @Override
964     public boolean isParameterConnected(DataFlowPanel containingPanel,
965         DataPartParameter dataPartParameter) {
966         ControlPart cp = getTabbedPanelControlPart(containingPanel);
967         if (cp == null)
968             return false;
969         for (ControlArrow a : controlArrows) {
970             if (a.getEndControlPart() == cp &&
971                 a.getArgumentConnections().containsKey(dataPartParameter.getName())) {
972                 return true;
973             }
974         }
975         return false;
976     }
977
978     @Override
979     public void parameterTypeChanged(DataFlowPanel containingPanel,
980         DataPartParameter dataPartParameter) {
981         ControlPart cp = getTabbedPanelControlPart(containingPanel);
982         if (cp == null)
983             return;
984         for (ControlArrow a : controlArrows) {
985             if (a.getEndControlPart() == cp &&
986                 a.getArgumentConnections().containsKey(dataPartParameter.getName())) {
987                 ControlPartTypeConnection conn = a.removeArgumentConnection(dataPartParameter.
988                     getName());
989                 ControlPartTypeConnection newConn = new ControlPartTypeConnection(conn.
990                     getOutputPart(),
991                     conn.getOutputName(), conn.getInputPart(), conn.getInputName(), true);
992                 a.putArgumentConnection(newConn);
993             }
994         }
995     }
996     @Override
997     public void changeParameterName(DataFlowPanel containingPanel,
```

```
997         DataPartParameter dataPartParameter, String oldName) {
998     ControlPart cp = getTabbedPanelControlPart(containingPanel);
999     if (cp == null)
1000         return;
1001     for (ControlArrow a : controlArrows) {
1002         if (a.getEndControlPart() == cp &&
1003             a.getArgumentConnections().containsKey(oldName)) {
1004             ControlPartTypeConnection conn = a.removeArgumentConnection(oldName);
1005             conn.setInputName(dataPartParameter.getName());
1006             a.putArgumentConnection(conn);
1007         }
1008     }
1009 }
1010
1011 @Override
1012 public void changeResultName(DataFlowPanel containingPanel,
1013     DataPartResult dataPartResult, String oldName) {
1014     ControlPart cp = getTabbedPanelControlPart(containingPanel);
1015     if (cp == null)
1016         return;
1017     for (ControlArrow a : controlArrows) {
1018         if (a.getStartControlPart() == cp) {
1019             Map<String,ControlPartTypeConnection> conns = a.getArgumentConnections();
1020             Collection<ControlPartTypeConnection> values = conns.values();
1021             for (ControlPartTypeConnection conn : values) {
1022                 if (conn.getOutputName().equals(oldName)) {
1023                     a.removeArgumentConnection(conn.getInputName());
1024                     conn.setOutputName(dataPartResult.getName());
1025                     a.putArgumentConnection(conn);
1026                 }
1027             }
1028         }
1029     }
1030 }
1031
1032 @Override
1033 public boolean isResultConnected(DataFlowPanel containingPanel,
1034     DataPartResult dataPartResult) {
1035     ControlPart cp = getTabbedPanelControlPart(containingPanel);
1036     if (cp == null)
1037         return false;
1038     for (ControlArrow a : controlArrows) {
1039         if (a.getStartControlPart() == cp) {
1040             Map<String,ControlPartTypeConnection> conns = a.getArgumentConnections();
1041             Collection<ControlPartTypeConnection> values = conns.values();
1042             for (ControlPartTypeConnection conn : values) {
1043                 if (conn.getOutputName().equals(dataPartResult.getName())) {
1044                     return true;
1045                 }
1046             }
1047         }
1048     }
1049     return false;
1050 }
1051
```

```
1052     @Override
1053     public void resultTypeChanged(DataFlowPanel containingPanel,
1054         DataPartResult dataPartResult) {
1055         ControlPart cp = getTabbedPaneControlPart(containingPanel);
1056         if (cp == null)
1057             return;
1058         for (ControlArrow a : controlArrows) {
1059             if (a.getStartControlPart() == cp) {
1060                 Map<String,ControlPartTypeConnection> conns = a.getArgumentConnections();
1061                 Collection<ControlPartTypeConnection> values = conns.values();
1062                 for (ControlPartTypeConnection conn : values) {
1063                     if (conn.getOutputName().equals(dataPartResult.getName())) {
1064                         a.removeArgumentConnection(conn.getInputName());
1065                         ControlPartTypeConnection newConn = new ControlPartTypeConnection(conn.
1066                             getOutputPart(),
1067                             conn.getOutputName(), conn.getInputPart(), conn.getInputName(),
1068                             true);
1069                         a.putArgumentConnection(newConn);
1070                     }
1071                 }
1072             }
1073         }
1074         public void resetCanFail() {
1075             childFailStateChanged();
1076         }
1077
1078     @Override
1079     public void childFailStateChanged() {
1080         for (ControlPart cp : controlParts) {
1081             cp.resetCanFail();
1082         }
1083     }
1084
1085     private ControlPart getSequenceNumberControlPart(int seq) {
1086         for (ControlPart p : controlParts) {
1087             if (p.getSequenceNumber() == seq) {
1088                 return p;
1089             }
1090         }
1091         return null;
1092     }
1093
1094     @Override
1095     public void scrollToComponent(int seqNumber) {
1096         ControlPart p = getSequenceNumberControlPart(seqNumber);
1097         if (p == null) {
1098             JOptionPane.showMessageDialog(mainWindow, "Cannot find component", "Unable to scroll",
1099                                         JOptionPane.INFORMATION_MESSAGE);
1100         }
1101         selectedComponentReset(true);
1102         selectedControlParts.clear();
1103         selectedControlParts.add(p);
```

```

1104         p.setDrawSelectedBorder(true);
1105         scrollToPoint(p.getLocation());
1106         repaint();
1107     }
1108
1109     @Override
1110     public void scrollToPath(int startSeq, int eventIndex, int endSeq) {
1111         for (ControlArrow a : controlArrows) {
1112             if (a.getStartControlPart().getSequenceNumber() == startSeq &&
1113                 a.getEndControlPart().getSequenceNumber() == endSeq &&
1114                 a.getStartCircleIndex() == eventIndex) {
1115                 selectedComponentReset(true);
1116                 selectedArrow = a;
1117                 a.setSelected(true);
1118                 scrollToPoint(a.getStartControlPart().getLocation());
1119                 repaint();
1120                 return;
1121             }
1122         }
1123         JOptionPane.showMessageDialog(mainWindow, "Cannot find edge",
1124             "Unable to scroll", JOptionPane.INFORMATION_MESSAGE);
1125     }
1126 }
```

LISTING E.103: gui/ControlFlowPanel.java

```

1 package gui.controlflow;
2
3 import java.util.ArrayList;
4 import java.util.Collection;
5 import java.util.HashMap;
6 import java.util.HashSet;
7 import java.util.List;
8 import java.util.Map;
9 import java.util.Set;
10
11 public class GraphUtil {
12     public static class Dominator {
13         private boolean errorPathDominates;
14         private ControlPart controlPart;
15
16         private Dominator(boolean errorPath, ControlPart c) {
17             errorPathDominates = errorPath;
18             controlPart = c;
19         }
20
21         public ControlPart getControlPart() {
22             return controlPart;
23         }
24
25         public boolean isErrorPathDominator() {
26             return errorPathDominates;
27         }
28
29     }
30 }
```

```
30     public boolean equals(Object oth) {
31         Dominator d = (Dominator)oth;
32         return d.isErrorPathDominator() == isErrorPathDominator() &&
33             d.getControlPart() == getControlPart();
34     }
35
36     @Override
37     public int hashCode() {
38         return controlPart.hashCode() ^ new Boolean(errorPathDominates).hashCode();
39     }
40 }
41
42     public static Map<ControlPart,List<ControlArrow>> getInputArrowMap(
43         List<ControlPart> controlParts, List<ControlArrow> arrows) {
44         Map<ControlPart,List<ControlArrow>> ret = new HashMap<>();
45         for (ControlPart p : controlParts) {
46             ret.put(p, new ArrayList<>());
47         }
48         for (ControlArrow a : arrows) {
49             ret.get(a.getEndControlPart()).add(a);
50         }
51         return ret;
52     }
53
54     public static Set<ControlPart> getDominatorControlPartSet(ControlPart pred, ControlPart part,
55         List<ControlPart> controlParts, List<ControlArrow> arrows) {
56         List<Dominator> doms = getDominatorControlPartSet(pred, part, controlParts, arrows);
57         HashSet<ControlPart> ret = new HashSet<>();
58         for (Dominator d : doms) {
59             ret.add(d.getControlPart());
60         }
61         return ret;
62     }
63
64     private static Set<ControlPart> getDominatorControlPartSet(ControlPart pred, ControlPart part,
65         List<ControlPart> controlParts, Map<ControlPart,List<ControlArrow>> aMap) {
66         List<Dominator> doms = getDominatorControlPartSet(pred, part, controlParts, aMap);
67         HashSet<ControlPart> ret = new HashSet<>();
68         for (Dominator d : doms) {
69             ret.add(d.getControlPart());
70         }
71         return ret;
72     }
73
74     // Get dominators of pred, including pred itself. pred must be predecessor to part.
75     public static List<Dominator> getDominatorControlPartSet(ControlPart pred, ControlPart part,
76         List<ControlPart> controlParts, List<ControlArrow> arrows) {
77         return getDominatorControlPartSet(pred, part, controlParts, getInputArrowMap(controlParts, arrows));
78     }
79
80     // Get dominators of pred, including pred itself. pred must be predecessor to part.
81     private static List<Dominator> getDominatorControlPartSet(ControlPart pred, ControlPart part,
82         List<ControlPart> controlParts, Map<ControlPart,List<ControlArrow>> aMap) {
83         List<Dominator> ret = new ArrayList<>();
84         Set<ControlPart> visited = new HashSet<>();
```

```
85
86     List<ControlArrow> partArrows = aMap.get(part);
87     Dominator initial = null;
88     for (ControlArrow in : partArrows) {
89         if (in.getStartControlPart() == pred) {
90             if (in.isErrorArrow()) {
91                 if (initial == null) {
92                     initial = new Dominator(true, pred);
93                 } else if (!initial.isErrorPathDominator()) {
94                     initial = null;
95                     break;
96                 }
97             } else {
98                 if (initial == null) {
99                     initial = new Dominator(false, pred);
100                } else if (initial.isErrorPathDominator()) {
101                    initial = null;
102                    break;
103                }
104            }
105        }
106    }
107    if (initial != null) {
108        ret.add(initial);
109    }
110
111    for (ControlPart test : controlParts) {
112        if (test == pred)
113            continue;
114        visited.clear();
115        if (isDominator(test, pred, aMap, visited, true))
116            ret.add(new Dominator(true, test));
117        if (isDominator(test, pred, aMap, visited, false))
118            ret.add(new Dominator(false, test));
119    }
120    return ret;
121 }
122
123 // Returns whether dominator dominates part. The only paths to part goes through dominator!
124 private static boolean isDominator(ControlPart dominator, ControlPart part,
125     Map<ControlPart,List<ControlArrow>> arrows, Set<ControlPart> visited, boolean errorDom
126 ) {
127     if (visited.contains(part)) {
128         return true;
129     }
130     visited.add(part);
131     List<ControlArrow> inArrows = arrows.get(part);
132     boolean ret = inArrows.size() > 0;
133     for (ControlArrow a : inArrows) {
134         ControlPart pred = a.getStartControlPart();
135         if ((dominator != pred || a.isErrorArrow() != errorDom) &&
136             !isDominator(dominator, pred, arrows, visited, errorDom)) {
137             ret = false;
138             break;
139         }
140     }
141 }
```

```
139         }
140     }
141
142     visited.remove(part);
143     return ret;
144 }
145
146 public static List<ControlArrow> getDeleteAffectedArrows(ControlArrow deleteArrow,
147     List<ControlPart> controlParts, List<ControlArrow> arrows) {
148     Map<ControlPart,List<ControlArrow>> aMap = getInputArrowMap(controlParts, arrows);
149     for (List<ControlArrow> as : aMap.values()) {
150         as.remove(deleteArrow);
151     }
152
153     List<ControlArrow> ret = new ArrayList<>();
154     for (ControlPart test : controlParts) {
155         ret.addAll(getBrokenArgumentArrows(test, controlParts, aMap));
156     }
157     return ret;
158 }
159
160 public static List<ControlArrow> getAllBrokenArgumentArrows(List<ControlPart> controlParts,
161     List<ControlArrow> arrows) {
162     Map<ControlPart,List<ControlArrow>> aMap = getInputArrowMap(controlParts, arrows);
163     List<ControlArrow> ret = new ArrayList<>();
164     for (ControlPart test : controlParts) {
165         ret.addAll(getBrokenArgumentArrows(test, controlParts, aMap));
166     }
167     return ret;
168 }
169
170 private static List<ControlArrow> getBrokenArgumentArrows(ControlPart part,
171     List<ControlPart> controlParts, Map<ControlPart,List<ControlArrow>> aMap) {
172     List<ControlArrow> ret = new ArrayList<>();
173
174     List<ControlArrow> inputs = aMap.get(part);
175     for (ControlArrow a : inputs) {
176         Set<ControlPart> doms = getDominatorControlPartSet(a.getStartControlPart(), part,
controlParts, aMap);
177         Collection<ControlPartTypeConnection> conns = a.getArgumentConnections().values();
178         for (ControlPartTypeConnection conn : conns) {
179             if (!doms.contains(conn.getOutputPart())) {
180                 ret.add(a);
181                 break;
182             }
183         }
184     }
185
186     return ret;
187 }
188 }
```

LISTING E.104: gui/controlflow/GraphUtil.java

```
1 package gui.controlflow;
2
3 import java.awt.BorderLayout;
4 import java.awt.Color;
5 import java.awt.Dimension;
6 import java.awt.FlowLayout;
7 import java.awt.GridBagConstraints;
8 import java.awt.GridBagLayout;
9 import java.awt.GridLayout;
10 import java.awt.event.ActionEvent;
11 import java.awt.event.ActionListener;
12 import java.awt.event.WindowAdapter;
13 import java.awt.event.WindowEvent;
14 import java.util.List;
15 import java.util.Map;
16
17 import javax.swing.BorderFactory;
18 import javax.swing.JDialog;
19 import javax.swing.JLabel;
20 import javax.swing.JOptionPane;
21 import javax.swing.JPanel;
22 import javax.swing.JScrollPane;
23 import javax.swing.border.Border;
24
25 import gui.ControlFlowPanel;
26 import gui.MainWindow;
27 import gui.ShowTypePanel;
28 import gui.TypeConnectionDialog;
29 import lang.type.RecordColumn;
30 import util.UtilButton;
31 import util.UtilFont;
32 import util.UtilScrollPane;
33
34 @SuppressWarnings("serial")
35 public class TypeMapper extends JDialog {
36     public TypeMapper(MainWindow mainWindow, ControlFlowPanel controlFlowPanel,
37             ControlArrow arrow) {
38         super(mainWindow);
39
40         UtilScrollPane scroll = new UtilScrollPane();
41         TypeMapperPanel mapperPanel = new TypeMapperPanel(arrow,
42                 controlFlowPanel, this, arrow.getArgumentConnections(), scroll);
43         if (mapperPanel.getRowsHeight() >= 430)
44             setSize(new Dimension(500, 500));
45         else
46             setSize(new Dimension(500, mapperPanel.getRowsHeight() + 70));
47
48         scroll.setViewport().add(mapperPanel);
49         getContentPane().add(scroll);
50         setLocationRelativeTo(mainWindow);
51         setModal(true);
52
53         setDefaultCloseOperation(JDialog.DO_NOTHING_ON_CLOSE);
54         addWindowListener(new WindowAdapter() {
```

```
55     @Override
56     public void windowClosing(WindowEvent e) {
57         if (!arrow.areDefinedArgumentsConnected()) {
58             int confirm = JOptionPane.showOptionDialog(TypeMapper.this,
59                     "Not all data inputs have been connected.\nClose window anyway?",
60                     "Confirmation", JOptionPane.YES_NO_OPTION,
61                     JOptionPane.QUESTION_MESSAGE, null, null, null);
62             if (confirm != 0) {
63                 return;
64             }
65             arrow.removeUnconnected();
66         }
67         controlFlowPanel.leaveSelectionState();
68         TypeMapper.this.dispose();
69     }
70 });
71
72     setVisible(true);
73 }
74 }
75
76 @SuppressWarnings("serial")
77 class TypeMapperPanel extends JPanel {
78     private int rowsHeight = 0;
79     private ControlFlowPanel controlFlowPanel;
80     private TypeMapper typeMapper;
81     private ControlPart controlPart;
82     private ControlArrow controlArrow;
83     private TypeConnectionDialog typeConnectionDialog;
84
85     TypeMapperPanel(ControlArrow arrow, ControlFlowPanel controlFlowPanel,
86                     TypeMapper typeMapper, Map<String,ControlPartTypeConnection> outMap,
87                     JScrollPane scroll) {
88         this.controlArrow = arrow;
89         this.controlPart = arrow.getEndControlPart();
90         this.controlFlowPanel = controlFlowPanel;
91         this.typeMapper = typeMapper;
92         this.typeConnectionDialog = new TypeConnectionDialog(typeMapper, scroll);
93
94         List<RecordColumn> rows = controlPart.getInputs();
95
96         GridBagLayout layout = new GridBagLayout();
97         GridBagConstraints constraints = new GridBagConstraints();
98         setLayout(layout);
99
100        for (int y = 0; y < rows.size(); y++) {
101            RecordColumn r = rows.get(y);
102
103            constraints.fill = GridBagConstraints.BOTH;
104            constraints.gridx = 0;
105            constraints.gridy = y;
106            constraints.weightx = 0;
107            constraints.weighty = 0;
108            JPanel left = getLeftSide(r);
109            add(left, constraints);
```

```
110
111     constraints.gridx = 1;
112     constraints.fill = GridBagConstraints.BOTH;
113     constraints.weightx = 100;
114     constraints.weighty = 100;
115     JPanel right = getRightSide(r, outMap.get(r.getId()));
116     add(right, constraints);
117
118     rowsHeight += Math.max(left.getPreferredSize().height, right.getPreferredSize().height
119 );
120 }
121
122     setBorder(BorderFactory.createMatteBorder(1, 1, 0, 1, Color.black));
123 }
124
125     public int getRowsHeight() {
126         return rowsHeight;
127     }
128
129     public JPanel getLeftSide(RecordColumn row) {
130         Border loweredBevel = BorderFactory.createLoweredBevelBorder();
131
132         JPanel left = new JPanel(new GridLayout(2,2));
133         JLabel outputLabel = new JLabel("Output");
134         outputLabel.setFont(UtilFont.textField);
135         outputLabel.setBorder(loweredBevel);
136         left.add(outputLabel);
137         UtilButton select = new UtilButton("Select");
138         select.addActionListener(new ActionListener() {
139             @Override
140             public void actionPerformed(ActionEvent e) {
141                 controlFlowPanel.enterSelectionState(controlArrow, row.getId());
142                 typeMapper.dispose();
143             }
144         });
145         JPanel selectPanel = new JPanel(new FlowLayout(FlowLayout.LEADING, 2, 2));
146         selectPanel.setBorder(loweredBevel);
147         selectPanel.add(select);
148         left.add(selectPanel);
149         JLabel inputLabel = new JLabel("Input");
150         inputLabel.setFont(UtilFont.textField);
151         inputLabel.setBorder(loweredBevel);
152         left.add(inputLabel);
153         JLabel nameLabel = new JLabel(row.getId());
154         nameLabel.setBorder(loweredBevel);
155         nameLabel.setFont(UtilFont.textField);
156         left.add(nameLabel);
157
158         JPanel outerLeft = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));
159         outerLeft.add(left);
160
161         JPanel ret = new JPanel(new GridBagLayout());
162         ret.add(outerLeft);
163
164         ret.setBorder(BorderFactory.createMatteBorder(0, 0, 1, 0, Color.black));
```

```

164     return ret;
165 }
166
167 public JPanel getRightSide(RecordColumn row, ControlPartTypeConnection outType) {
168     JPanel right = new JPanel();
169
170     if (outType == null) {
171         right.setLayout(new GridBagLayout());
172         right.setBackground(Color.white);
173         right.setBorder(BorderFactory.createMatteBorder(0, 1, 1, 0, Color.black));
174         right.add(new ShowTypePanel(row.getType(), false));
175     } else {
176         right.setLayout(new BorderLayout());
177         right.setBackground(Color.white);
178         right.setBorder(BorderFactory.createMatteBorder(0, 1, 1, 0, Color.black));
179
180         JPanel panel = typeConnectionDialog.getConnectionPanel(outType);
181         String title = outType.getOutputPart().getTitle() + " (" + outType.getOutputName() + "
182         )";
183         //JPanel container = new JPanel(new BorderLayout());
184         JPanel top = new JPanel(new FlowLayout(FlowLayout.CENTER, 0, 0));
185         top.add(new JLabel(title));
186         right.add(top, BorderLayout.NORTH);
187         right.add(panel);
188         //right.add(container);
189     }
190     return right;
191 }
```

LISTING E.105: gui/controlflow/TypeMapper.java

```

1 package gui.controlflow.box;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import gui.ControlFlowPanel;
7 import gui.TabbedPane;
8 import gui.controlflow.ControlPartType;
9 import lang.type.RecordColumn;
10
11 public class InteractionBox implements Box {
12
13     private String title;
14     private String[] events;
15     private List<RecordColumn> inputs;
16     private List<RecordColumn> outputs;
17     private String platform;
18
19     public InteractionBox(String title, String platform, String[] events, ControlFlowPanel cfp,
20             List<RecordColumn> inputs, List<RecordColumn> outputs) {
21         this.title = title;
22         this.events = events.clone();
23         this.inputs = new ArrayList<>(inputs);
```

```
24     this.outputs = new ArrayList<>(outputs);
25     this.platform = platform;
26 }
27
28 @Override
29 public String getPlatform() {
30     return platform;
31 }
32
33 @Override
34 public String getTitle() {
35     return title;
36 }
37
38 @Override
39 public String[] getEvents() {
40     return events.clone();
41 }
42
43 @Override
44 public boolean canFail() {
45     return true;
46 }
47
48 @Override
49 public TabbedPane getTabbedPane() {
50     return null;
51 }
52
53 @Override
54 public List<RecordColumn> getInputs() {
55     return new ArrayList<>(inputs);
56 }
57
58 @Override
59 public List<RecordColumn> getOutputs() {
60     return new ArrayList<>(outputs);
61 }
62
63 @Override
64 public ControlPartType getType() {
65     return ControlPartType.INTERACTION;
66 }
67
68 }
```

LISTING E.106: gui/controlflow/box/InteractionBox.java

```
1 package gui.controlflow.box;
2
3 import gui.TabbedPane;
4 import gui.controlflow.ControlPartType;
5
6 import java.util.ArrayList;
7 import java.util.List;
```

```

8
9 import lang.type.RecordColumn;
10 import lang.type.TypeBool;
11
12 public class IfRuntimeInteractionBox implements Box {
13     private final static ArrayList<RecordColumn> inputs = new ArrayList<>();
14     static {
15         inputs.add(new RecordColumn("Condition", new TypeBool()));
16     }
17
18     private final static String[] events = {"True", "False"};
19     @Override
20     public String getTitle() {
21         return "If";
22     }
23     @Override
24     public String[] getEvents() {
25         return events.clone();
26     }
27     @Override
28     public boolean canFail() {
29         return false;
30     }
31     @Override
32     public TabbedPane getTabbedPane() {
33         return null;
34     }
35     @Override
36     public List<RecordColumn> getInputs() {
37         return new ArrayList<>(inputs);
38     }
39     @Override
40     public List<RecordColumn> getOutputs() {
41         return new ArrayList<>();
42     }
43
44     @Override
45     public ControlPartType getType() {
46         return ControlPartType.IF;
47     }
48 }
```

LISTING E.107: gui/controlflow/box/IfRuntimeInteractionBox.java

```

1 package gui.controlflow.box;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import gui.TabbedPane;
7 import gui.controlflow.ControlPartType;
8 import lang.type.RecordColumn;
9 import lang.type.TypeError;
10
11 public class FailBox implements Box {
```

```

12     @Override
13     public String getTitle() {
14         return "Fail";
15     }
16
17     @Override
18     public String[] getEvents() {
19         return new String[]{};
20     }
21
22     @Override
23     public boolean canFail() {
24         return false;
25     }
26
27     @Override
28     public TabbedPane getTabbedPane() {
29         return null;
30     }
31
32     @Override
33     public List<RecordColumn> getInputs() {
34         ArrayList<RecordColumn> in = new ArrayList<>();
35         in.add(new RecordColumn("Error", new TypeError()));
36         return in;
37     }
38
39     @Override
40     public List<RecordColumn> getOutputs() {
41         return new ArrayList<>();
42     }
43
44     @Override
45     public ControlPartType getType() {
46         return ControlPartType.FAIL;
47     }
48 }
```

LISTING E.108: gui/controlflow/box/FailBox.java

```

1 package gui.controlflow.box;
2
3 import gui.ControlFlowPanel;
4 import gui.DataFlowPanel;
5 import gui.MainWindow;
6 import gui.TabbedPane;
7 import gui.TabbedPane.InvalidTitleException;
8 import gui.controlflow.ControlPartType;
9
10 import java.util.List;
11
12 import lang.type.RecordColumn;
13
14 public class DataFlowBox implements Box {
15     private DataFlowPanel dataFlowPanel;
```

```

16     private String[] events = {"Success"};
17
18     public DataFlowBox(MainWindow mainWindow, String title, ControlFlowPanel cfp)
19         throws InvalidTitleException {
20         dataFlowPanel = new DataFlowPanel(mainWindow, title, cfp);
21     }
22
23     @Override
24     public String getTitle() {
25         return dataFlowPanel.getTitle();
26     }
27
28     @Override
29     public String[] getEvents() {
30         return events.clone();
31     }
32
33     @Override
34     public boolean canFail() {
35         return dataFlowPanel.canFail();
36     }
37
38     @Override
39     public TabbedPane getTabbedPanel() {
40         return dataFlowPanel;
41     }
42
43     @Override
44     public List<RecordColumn> getInputs() {
45         return dataFlowPanel getArguments();
46     }
47
48     @Override
49     public List<RecordColumn> getOutputs() {
50         return dataFlowPanel.getResults();
51     }
52
53     @Override
54     public ControlPartType getType() {
55         return ControlPartType.DATAFLOW;
56     }
57 }
```

LISTING E.109: gui/controlflow/box/DataFlowBox.java

```

1 package gui.controlflow.box;
2
3 import java.util.List;
4
5 import lang.type.RecordColumn;
6 import gui.TabbedPane;
7 import gui.controlflow.ControlPartType;
8
9 public interface Box {
10     String getTitle();
```

```
11     String[] getEvents();
12     boolean canFail();
13     TabbedPanel getTabbedPanel();
14     List<RecordColumn> getInputs();
15     List<RecordColumn> getOutputs();
16     ControlPartType getType();
17     default String getPlatform() { return null; }
18 }
```

LISTING E.110: gui/controlflow/box/Box.java

```
1 package gui.controlflow.box;
2
3 import gui.TabbedPanel;
4 import gui.controlflow.ControlPartType;
5
6 import java.util.ArrayList;
7 import java.util.List;
8
9 import lang.type.RecordColumn;
10 import lang.type.TypeString;
11
12 public class StartBox implements Box {
13     private static final ArrayList<RecordColumn> outputs = new ArrayList<>();
14     private static final String[] event = {"OK"};
15     static {
16         outputs.add(new RecordColumn("Username", new TypeString()));
17     }
18
19     @Override
20     public String getTitle() {
21         return "Start";
22     }
23     @Override
24     public String[] getEvents() {
25         return event.clone();
26     }
27
28     @Override
29     public boolean canFail() {
30         return false;
31     }
32
33     @Override
34     public TabbedPanel getTabbedPanel() {
35         return null;
36     }
37
38     @Override
39     public List<RecordColumn> getInputs() {
40         return new ArrayList<>();
41     }
42
43     @Override
44     public List<RecordColumn> getOutputs() {
```

```
45         return new ArrayList<>(outputs);
46     }
47
48     @Override
49     public ControlPartType getType() {
50         return ControlPartType.START;
51     }
52 }
```

LISTING E.111: gui/controlflow/box/StartBox.java

```
1 package gui.controlflow.box;
2
3 import gui.TabbedPane;
4 import gui.controlflow.ControlPartType;
5
6 import java.util.ArrayList;
7 import java.util.List;
8
9 import lang.type.RecordColumn;
10
11 public class StartBox implements Box {
12     @Override
13     public String getTitle() {
14         return "Start";
15     }
16
17     @Override
18     public String[] getEvents() {
19         return new String[]{};
20     }
21
22     @Override
23     public boolean canFail() {
24         return false;
25     }
26
27     @Override
28     public TabbedPane getTabbedPane() {
29         return null;
30     }
31
32     @Override
33     public List<RecordColumn> getInputs() {
34         return new ArrayList<>();
35     }
36
37     @Override
38     public List<RecordColumn> getOutputs() {
39         return new ArrayList<>();
40     }
41
42     @Override
43     public ControlPartType getType() {
44         return ControlPartType.START;
```

```
45     }
46 }
```

LISTING E.112: gui/controlflow/box/StopBox.java

```
1 package gui.controlflow;
2
3 import java.awt.BasicStroke;
4 import java.awt.Color;
5 import java.awt.Dimension;
6 import java.awt.Graphics2D;
7 import java.awt.Point;
8 import java.awt.Rectangle;
9 import java.awt.Stroke;
10 import java.awt.geom.Line2D;
11 import java.util.ArrayList;
12 import java.util.HashMap;
13 import java.util.List;
14 import java.util.Map;
15
16 import gui.ControlFlowPanel;
17 import gui.DataFlowPanel;
18 import gui.MainWindow;
19 import gui.dataflow.DataPartResult;
20 import lang.type.RecordColumn;
21 import lang.type.Type;
22 import util.ColorTheme;
23
24 public class ControlArrow {
25     private int startCircleIndex;
26     private ControlPart startControlPart;
27     private ControlPart endControlPart;
28     private Point fixedEndPoint;
29     private boolean selected;
30     private boolean hovered;
31     private Map<String,ControlPartTypeConnection> argumentConnections = new HashMap<>();
32
33     public ControlArrow(ControlPart start, int startCircleIndex, ControlPart end) {
34         this.startCircleIndex = startCircleIndex;
35         startControlPart = start;
36         endControlPart = end;
37     }
38
39     public ControlArrow(ControlPart start, int startCircleIndex, Point end) {
40         this.startCircleIndex = startCircleIndex;
41         startControlPart = start;
42         fixedEndPoint = new Point(end.x, end.y);
43     }
44
45     public boolean isErrorArrow() {
46         return startControlPart.isErrorCircle(startCircleIndex);
47     }
48
49     public ControlPart getStartControlPart() {
50         return startControlPart;
```

```
51     }
52
53     public ControlPart getEndControlPart() {
54         return endControlPart;
55     }
56
57     public int getStartCircleIndex() {
58         return startCircleIndex;
59     }
60
61     public void setHovered(boolean hovered) {
62         this.hovered = hovered;
63     }
64
65     public void setSelected(boolean selected) {
66         this.selected = selected;
67     }
68
69     public boolean intersects(Point p) {
70         LinePoints ps = getLinePoints();
71         Rectangle r = new Rectangle(p.x-3,p.y-3,6,6);
72
73         Line2D[] lines;
74         if (ps.middle != null) {
75             lines = new Line2D[4];
76             lines[0] = new Line2D.Double(ps.start.x, ps.start.y, ps.middle1.x, ps.middle1.y);
77             lines[1] = new Line2D.Double(ps.middle1.x, ps.middle1.y, ps.middle.x, ps.middle.y);
78             lines[2] = new Line2D.Double(ps.middle.x, ps.middle.y, ps.middle2.x, ps.middle2.y);
79             lines[3] = new Line2D.Double(ps.middle2.x, ps.middle2.y, ps.end.x, ps.end.y);
80         } else {
81             lines = new Line2D[2];
82             lines[0] = new Line2D.Double(ps.start.x, ps.start.y, ps.end.x, ps.start.y);
83             lines[1] = new Line2D.Double(ps.end.x, ps.start.y, ps.end.x, ps.end.y);
84         }
85         for (Line2D line : lines) {
86             if (line.intersects(r))
87                 return true;
88         }
89         return false;
90     }
91
92     public void paint(Graphics2D g2) {
93         LinePoints ps = getLinePoints();
94         drawBaseLine(g2, ps);
95         drawHoveredLine(g2, ps);
96         drawSelectedLine(g2, ps);
97     }
98
99     private void drawBaseLine(Graphics2D g2, LinePoints ps) {
100         Color c;
101         //if (startControlPart.isErrorCircle(startCircleIndex))
102         if (areAllArgumentsConnected())
103             c = ColorTheme.DARK_GREEN;
104         else
105             c = ColorTheme.DARK_RED;
```

```
106         drawLine(g2, ps, c, 2);
107     }
108
109     private void drawHoveredLine(Graphics2D g2, LinePoints ps) {
110         if (hovered)
111             drawLine(g2, ps, ColorTheme.HOVERED, 6);
112     }
113
114     private void drawSelectedLine(Graphics2D g2, LinePoints ps) {
115         if (selected)
116             drawLine(g2, ps, ColorTheme.SELECTED, 4);
117     }
118
119     private void drawMovedLine(Graphics2D g2, Point start, Point end, int lineWidth) {
120         if (start.x < end.x) {
121             start = new Point(start.x+lineWidth, start.y);
122         } else if (start.y < end.y) {
123             start = new Point(start.x, start.y+lineWidth);
124         } else if (start.x > end.x) {
125             start = new Point(start.x-lineWidth, start.y);
126         } else if (start.y > end.y) {
127             start = new Point(start.x, start.y-lineWidth);
128         } else {
129             return;
130         }
131         drawSingleLine(g2, start, end, lineWidth);
132     }
133
134     private void drawSingleLine(Graphics2D g2, Point start, Point end, int lineWidth) {
135         g2.drawLine(start.x, start.y, end.x, end.y);
136     }
137
138     private void drawLine(Graphics2D g2, LinePoints ps, Color lineColor, int lineWidth) {
139         Stroke orig = g2.getStroke();
140         g2.setStroke(new BasicStroke(lineWidth));
141         g2.setColor(lineColor);
142         if (ps.middle != null) {
143             drawSingleLine(g2, ps.start, ps.middle1, lineWidth);
144             drawMovedLine(g2, ps.middle1, ps.middle, lineWidth);
145             drawMovedLine(g2, ps.middle, ps.middle2, lineWidth);
146             if (ps.downArrow)
147                 drawMovedLine(g2, ps.middle2, new Point(ps.end.x, ps.end.y-lineWidth), lineWidth);
148             else
149                 drawMovedLine(g2, ps.middle2, new Point(ps.end.x, ps.end.y+lineWidth), lineWidth);
150         } else {
151             drawSingleLine(g2, ps.start, new Point(ps.end.x, ps.start.y), lineWidth);
152             if (ps.downArrow)
153                 drawMovedLine(g2, new Point(ps.end.x, ps.start.y), new Point(ps.end.x, ps.end.y-
154                             lineWidth), lineWidth);
155             else
156                 drawMovedLine(g2, new Point(ps.end.x, ps.start.y), new Point(ps.end.x, ps.end.y+
157                             lineWidth), lineWidth);
158         }
159         g2.setColor(Color.black);
160         g2.setStroke(orig);
```

```
159     if(ps.downArrow)
160         g2.fillPolygon(new int[]{ps.end.x-5, ps.end.x+5, ps.end.x},
161                         new int[]{ps.end.y-7, ps.end.y-7, ps.end.y+3}, 3);
162     else
163         g2.fillPolygon(new int[]{ps.end.x-5, ps.end.x+5, ps.end.x},
164                         new int[]{ps.end.y+7, ps.end.y+7, ps.end.y-3}, 3);
165     }
166
167     private LinePoints getLinePoints() {
168         Point center = startControlPart.getCircleCenter(startCircleIndex);
169         Dimension endSize;
170         Point endLocation;
171         if (endControlPart != null) {
172             endSize = endControlPart.getSize();
173             endLocation = endControlPart.getLocation();
174         } else {
175             endSize = new Dimension(0,0);
176             endLocation = fixedEndPoint;
177         }
178
179         Point top = new Point(endLocation.x+endSize.width/2, endLocation.y);
180         Point bottom = new Point(top.x, endLocation.y+endSize.height);
181
182         Point start = center;
183         Point middle1;
184         Point middle = null;
185         Point middle2;
186         Point end;
187         boolean downArrow;
188         int sideIndex = startCircleIndex/2;
189
190         if (startControlPart.isLeftCircle(startCircleIndex)) {
191             start.x -= 1;
192
193             int extra = (1+sideIndex) * 10;
194             middle1 = new Point(start.x - extra, start.y);
195
196             if (fixedEndPoint == null) {
197                 if (top.distance(start) < bottom.distance(start)) {
198                     downArrow = true;
199                     end = new Point(top.x, top.y-3);
200                 } else {
201                     downArrow = false;
202                     end = new Point(bottom.x, bottom.y+3);
203                 }
204             } else {
205                 if (fixedEndPoint.y >= start.y) {
206                     downArrow = true;
207                 } else {
208                     downArrow = false;
209                 }
210                 end = new Point(fixedEndPoint.x, fixedEndPoint.y);
211             }
212             extra = 6 + (startControlPart.getNumberCircles() - sideIndex) * 10;
213         }
```

```
214         if (downArrow) {
215             middle2 = new Point(end.x, end.y-extra);
216             if (middle2.y < start.y || end.x >= start.x)
217                 middle = new Point(middle1.x, middle2.y);
218         } else {
219             middle2 = new Point(end.x, end.y+extra);
220             if (middle2.y > start.y || end.x >= start.x)
221                 middle = new Point(middle1.x, middle2.y);
222         }
223     } else {
224         start.x += 1;
225
226         int extra = (startControlPart.getNumberCircles() - sideIndex) * 10;
227         middle1 = new Point(start.x + extra, start.y);
228
229         if (fixedEndPoint == null) {
230             if (top.distance(start) <= bottom.distance(start)) {
231                 downArrow = true;
232                 end = new Point(top.x, top.y-3);
233             } else {
234                 downArrow = false;
235                 end = new Point(bottom.x, bottom.y+3);
236             }
237         } else {
238             if (fixedEndPoint.y > start.y) {
239                 downArrow = true;
240             } else {
241                 downArrow = false;
242             }
243             end = new Point(fixedEndPoint.x, fixedEndPoint.y);
244         }
245         extra = 6 + (1+sideIndex) * 10;
246         if (downArrow) {
247             middle2 = new Point(end.x, end.y-extra);
248             if (middle2.y < start.y || end.x <= start.x)
249                 middle = new Point(middle1.x, middle2.y);
250         } else {
251             middle2 = new Point(end.x, end.y+extra);
252             if (middle2.y > start.y || end.x <= start.x)
253                 middle = new Point(middle1.x, middle2.y);
254         }
255     }
256
257     if (middle != null && (middle1.distance(middle) < 6 || middle2.distance(middle) < 6))
258         middle = null;
259
260     LinePoints ret = new LinePoints();
261     ret.start = start;
262     ret.middle1 = middle1;
263     ret.middle = middle;
264     ret.middle2 = middle2;
265     ret.end = end;
266     ret.downArrow = downArrow;
267     return ret;
268 }
```

```
269
270     private static class LinePoints {
271         Point start;
272         Point middle1;
273         Point middle;
274         Point middle2;
275         Point end;
276         boolean downArrow;
277     }
278
279     public void openTypeMapper(MainWindow w, ControlFlowPanel cfp) {
280         if (getEndControlPart().getInputs().size() > 0)
281             new TypeMapper(w, cfp, this);
282     }
283
284     public void putArgumentConnection(ControlPartTypeConnection conn) {
285         String argName = conn.getInputName();
286
287         Map<String,Type> args = getEndControlPart().getInputMap();
288         if (!args.containsKey(argName)) {
289             throw new IllegalArgumentException("input '" + argName + "' does not exist");
290         }
291         argumentConnections.put(argName, conn);
292     }
293
294     public ControlPartTypeConnection removeArgumentConnection(String argName) {
295         ControlPartTypeConnection c = argumentConnections.remove(argName);
296         if (c == null)
297             throw new IllegalArgumentException("input '" + argName + "' does not exist");
298         notifyDataFlowAboutRemove(c.getOutputPart(), c.getOutputName());
299         return c;
300     }
301
302     private void notifyDataFlowAboutRemove(ControlPart cp, String name) {
303         if (!cp.isDataFlow())
304             return;
305         DataFlowPanel df = (DataFlowPanel)cp.getTabbedPanel();
306         DataPartResult res = df.getDataPartResult(name);
307         if (res.getType().hasAuto())
308             res.autoTypeOutputRemoved();
309     }
310
311     public boolean hasArgumentConnection(String name) {
312         return argumentConnections.containsKey(name);
313     }
314
315     public Map<String, ControlPartTypeConnection> getArgumentConnections() {
316         return new HashMap<>(argumentConnections);
317     }
318
319     public boolean areAllArgumentsConnected() {
320         if (getEndControlPart() == null)
321             return true;
322
323         List<RecordColumn> args = getEndControlPart().getInputs();
```

```

324         for (RecordColumn col : args) {
325             if (!argumentConnections.containsKey(col.getId()))
326                 return false;
327         }
328         return areDefinedArgumentsConnected();
329     }
330
331     public boolean areDefinedArgumentsConnected() {
332         List<RecordColumn> args = getEndControlPart().getInputs();
333         for (RecordColumn col : args) {
334             ControlPartTypeConnection conn = argumentConnections.get(col.getId());
335             if (conn == null)
336                 continue;
337             if (!conn.isConnected())
338                 return false;
339         }
340         return true;
341     }
342
343     public void removeUnconnected() {
344         ArrayList<String> keys = new ArrayList<>();
345         for (Map.Entry<String,ControlPartTypeConnection> e : argumentConnections.entrySet()) {
346             if (e.getValue().isConnected())
347                 continue;
348             keys.add(e.getKey());
349         }
350         for (String k : keys)
351             removeArgumentConnection(k);
352     }
353 }
```

LISTING E.113: gui/controlflow/ControlArrow.java

```

1 package gui.controlflow;
2
3 import gui.ControlFlowPanel;
4 import gui.MainWindow;
5 import gui.TabbedPane;
6 import gui.TabbedPane.InvalidTitleException;
7 import gui.controlflow.box.Box;
8 import gui.controlflow.box.DataFlowBox;
9 import gui.controlflow.box.FailBox;
10 import gui.controlflow.box.IfRuntimeInteractionBox;
11 import gui.controlflow.box.InteractionBox;
12 import gui.controlflow.box.StartBox;
13 import gui.controlflow.box.StopBox;
14 import lang.type.RecordColumn;
15 import lang.type.Type;
16 import lang.type.TypeError;
17
18 import java.awt.BasicStroke;
19 import java.awt.Color;
20 import java.awt.Dimension;
21 import java.awt.Font;
22 import java.awt.Graphics;
```

```
23 import java.awt.Graphics2D;
24 import java.awt.Point;
25 import java.awt.Rectangle;
26 import java.awt.Stroke;
27 import java.awt.font.TextLayout;
28 import java.awt.geom.Rectangle2D;
29 import java.awt.geom.RoundRectangle2D;
30 import java.util.ArrayList;
31 import java.util.Arrays;
32 import java.util.HashMap;
33 import java.util.List;
34 import java.util.Map;
35
36 import util.ColorTheme;
37 import util.UtilFont;
38
39 public class ControlPart implements TabbedPane.TitleChangedListener {
40     private Rectangle bounds;
41     private Box box;
42     private final static Font titleFont = UtilFont.titleFont;
43     private final static Font subTitleFont = UtilFont.smallItalicFont;
44     private static final Font eventFont = UtilFont.textFont;
45     private final static int extraSpace = 3;
46     private final static int titlesGap = 4;
47     private final static int titleSpace = 4*extraSpace;
48     private final static int circleDiameter = 10;
49
50     public final static String errorEvent = "Error";
51
52     private final static TypeError typeError = new TypeError();
53     public final static RecordColumn[] errorRecordColumn = {new RecordColumn(errorEvent, typeError
54     )};
55
56     private boolean drawFocusBorder = false;
57     private boolean drawSelectedBorder = false;
58     private String title;
59     private String subTitle;
60     private TextLayout titleLayout;
61     private TextLayout subTitleLayout;
62     private String[] events;
63     private TextLayout[] eventLayouts;
64     private Point[] relativeCircleLocations;
65     private int focusCircleIndex = -1;
66     private Graphics2D initialGraphics2D;
67     private int sequenceNumber = -1;
68
69     private ControlPart(ControlFlowPanel controlFlowPanel, Graphics g, String subTitle) {
70         this.subTitle = subTitle;
71         this.bounds = new Rectangle(0, 0, 0, 0);
72         initialGraphics2D = (Graphics2D)g;
73     }
74
75     public int getSequenceNumber() {
76         return sequenceNumber;
77     }
```

```
77
78     public void setSequenceNumber(int n) {
79         this.sequenceNumber = n;
80     }
81
82     public static ControlPart getInteractionControlPart(String title, String platform,
83             String[] events, List<RecordColumn> inputs, List<RecordColumn> outputs,
84             ControlFlowPanel cfp, Graphics g) {
85         ControlPart ret = new ControlPart(cfp, g, platform);
86         ret.setInteractionBox(title, platform, events, inputs, outputs, cfp);
87         ret.initialize(ret.initialGraphics2D);
88         return ret;
89     }
90
91     public static ControlPart getStartControlPart(ControlFlowPanel cfp, Graphics g) {
92         ControlPart ret = new ControlPart(cfp, g, "");
93         ret.setStartBox();
94         ret.initialize(ret.initialGraphics2D);
95         return ret;
96     }
97
98     public static ControlPart getFailControlPart(ControlFlowPanel cfp, Graphics g) {
99         ControlPart ret = new ControlPart(cfp, g, "");
100        ret.setFailBox();
101        ret.initialize(ret.initialGraphics2D);
102        return ret;
103    }
104
105    public static ControlPart getStopControlPart(ControlFlowPanel cfp, Graphics g) {
106        ControlPart ret = new ControlPart(cfp, g, "");
107        ret.setStopBox();
108        ret.initialize(ret.initialGraphics2D);
109        return ret;
110    }
111
112    public static ControlPart getDataflowControlPart(MainWindow win, String title,
113             ControlFlowPanel controlFlowPanel, Graphics g)
114             throws InvalidTitleException {
115        ControlPart ret = new ControlPart(controlFlowPanel, g, "Dataflow");
116        ret.setDataflowBox(win, title, controlFlowPanel);
117        ret.initialize(ret.initialGraphics2D);
118        return ret;
119    }
120
121    public static ControlPart getIfRuntimeInteractionControlPart(ControlFlowPanel controlFlowPanel
122             , Graphics g) {
123        ControlPart ret = new ControlPart(controlFlowPanel, g, "");
124        ret.setIfRuntimeInteractionBox();
125        ret.initialize(ret.initialGraphics2D);
126        return ret;
127    }
128
129    public TabbedPane getTabbedPane() {
130        return box.getTabbedPane();
131    }
```

```
131
132     private void setStartBox() {
133         box = new StartBox();
134     }
135
136     private void setStopBox() {
137         box = new StopBox();
138     }
139
140     private void setFailBox() {
141         box = new FailBox();
142     }
143
144     private void setInteractionBox(String title, String platform, String[] events, List<
145         RecordColumn> inputs,
146         List<RecordColumn> outputs, ControlFlowPanel cfp) {
147         box = new InteractionBox(title, platform, events, cfp, inputs, outputs);
148     }
149
150     private void setDataflowBox(MainWindow w, String title, ControlFlowPanel cfp)
151         throws InvalidTitleException {
152         box = new DataFlowBox(w, title, cfp);
153         box.getTabbedPane().addTitleChangedListener(this);
154     }
155
156     private void setIfRuntimeInteractionBox() {
157         box = new IfRuntimeInteractionBox();
158     }
159
160     public void setLocation(Point p){
161         bounds.x = Math.max(p.x, 0);
162         bounds.y = Math.max(p.y, 0);
163     }
164
165     public void moveLocation(Point delta) {
166         bounds.x += delta.x;
167         bounds.y += delta.y;
168         bounds.x = Math.max(bounds.x, 0);
169         bounds.y = Math.max(bounds.y, 0);
170     }
171
172     public Point getLocation(){
173         return new Point(bounds.x, bounds.y);
174     }
175
176     public void setDrawFocusBorder(boolean drawFocusBorder) {
177         this.drawFocusBorder = drawFocusBorder;
178     }
179
180     public void setDrawSelectedBorder(boolean drawSelectedBorder) {
181         this.drawSelectedBorder = drawSelectedBorder;
182     }
183
184     private void drawFocusRectangle(Graphics2D g2) {
185         if (!drawFocusBorder)
```

```
185         return;
186     RoundRectangle2D rect = new RoundRectangle2D.Double(bounds.x-1,bounds.y-1,bounds.width+2,
187     bounds.height+2,10,10);
188     Stroke d = g2.getStroke();
189     g2.setStroke(new BasicStroke(4));
190     g2.setColor(ColorTheme.HOVERED);
191     g2.draw(rect);
192     g2.setStroke(d);
193     g2.setColor(Color.BLACK);
194 }
195
196 private void drawSelectedRectangle(Graphics2D g2) {
197     if (!drawSelectedBorder)
198         return;
199     RoundRectangle2D rect = new RoundRectangle2D.Double(bounds.x-1,bounds.y-1,bounds.width+2,
200     bounds.height+2,10,10);
201     Stroke d = g2.getStroke();
202     g2.setStroke(new BasicStroke(3));
203     g2.setColor(ColorTheme.SELECTED);
204     g2.draw(rect);
205     g2.setStroke(d);
206     g2.setColor(Color.BLACK);
207 }
208
209 private void drawFocusCircle(Graphics2D g2) {
210     if (focusCircleIndex < 0)
211         return;
212     Point rel = relativeCircleLocations[focusCircleIndex];
213     g2.setColor(ColorTheme.HOVERED);
214     g2.fillOval(rel.x+bounds.x-3, rel.y+bounds.y-3, circleDiameter+6, circleDiameter+6);
215     g2.setColor(Color.black);
216 }
217
218 private void initialize(Graphics2D g2) {
219     title = box.getTitle();
220     if (title.equals(""))
221         title = " ";
222     if (subTitle.equals(""))
223         subTitle = " ";
224
225     titleLayout = new TextLayout(title, titleFont, g2.getFontRenderContext());
226     subTitleLayout = new TextLayout(subTitle, subTitleFont, g2.getFontRenderContext());
227
228     Rectangle2D titleBounds = titleLayout.getBounds();
229     Rectangle2D subTitleBounds = subTitleLayout.getBounds();
230
231     bounds.width = Math.max((int)Math.ceil(titleBounds.getWidth()), (int)Math.ceil(
232     subTitleBounds.getWidth()));
233     bounds.height = (int)Math.ceil(titleBounds.getHeight()) + (int)Math.ceil(subTitleBounds.
234     getHeight()) + titlesGap;
235     bounds.height += titleSpace + extraSpace;
236     bounds.width += 4*extraSpace;
237
238     events = box.getEvents();
239     if (box.canFail()) {
```

```
236     String[] old = events;
237     events = new String[events.length + 1];
238     for (int i = 0; i < old.length; i++)
239         events[i] = old[i];
240     events[old.length] = errorEvent;
241 }
242 relativeCircleLocations = new Point[events.length*2];
243 eventLayouts = new TextLayout[events.length];
244 for (int i = 0, relIdx = 0; i < events.length; i++, relIdx += 2) {
245
246     String event = events[i];
247     if (event.equals(""))
248         event = " ";
249     eventLayouts[i] = new TextLayout(event, eventFont,
250         g2.getFontRenderContext());
251     Rectangle2D eventBounds = eventLayouts[i].getBounds();
252     int eventStringWidth = (int) Math.ceil(eventBounds.getWidth());
253     int extraEventWidth = 4*extraSpace + circleDiameter;
254     bounds.width = Math.max(bounds.width, eventStringWidth + extraEventWidth);
255     int eventStringHeight = (int) Math.ceil(eventBounds.getHeight());
256     int circleY = bounds.height + (eventStringHeight+extraSpace*2)/2 - circleDiameter/2;
257     bounds.height += eventStringHeight;
258     bounds.height += extraSpace * 4;
259
260     relativeCircleLocations[relIdx] = new Point(-circleDiameter/2, circleY);
261     relativeCircleLocations[relIdx+1] = new Point(-circleDiameter/2, circleY);
262 }
263 for (int i = 1; i < relativeCircleLocations.length; i += 2) {
264     relativeCircleLocations[i].x += bounds.width;
265 }
266 bounds.height += 2*extraSpace;
267 }
268
269 public int getNumberCircles() {
270     return events.length;
271 }
272
273 public void paint(Graphics2D g2, boolean transparent){
274     drawBoundingRect(g2, transparent);
275     drawFocusRectangle(g2);
276     drawSelectedRectangle(g2);
277     int yLoc = drawTitle(g2) + extraSpace;
278     for (int i = 0; i < events.length; i++) {
279         yLoc = drawEvent(g2, events[i], eventLayouts[i].getBounds(), yLoc);
280     }
281     drawFocusCircle(g2);
282 }
283 private void drawCircles(Graphics2D g2, int circleY) {
284     /*
285     Polygon c1 = new Polygon();
286     //g2.fillOval(bounds.x-circleDiameter/2, circleY, circleDiameter, circleDiameter);
287     c1.addPoint(bounds.x+circleDiameter/2, circleY);
288     c1.addPoint(bounds.x+circleDiameter/2, circleY+circleDiameter);
289     c1.addPoint(bounds.x-circleDiameter/2, circleY+circleDiameter/2);
290     g2.fill(c1);
```

```
291
292     Polygon c2 = new Polygon();
293     //g2.fillOval(bounds.x+bounds.width-circleDiameter/2, circleY, circleDiameter,
294     //circleDiameter);
295     c2.addPoint(bounds.x+bounds.width-circleDiameter/2, circleY);
296     c2.addPoint(bounds.x+bounds.width-circleDiameter/2, circleY+circleDiameter);
297     c2.addPoint(bounds.x+bounds.width+circleDiameter/2, circleY+circleDiameter/2);
298     g2.fill(c2);
299     */
300 }
301 private int drawEvent(Graphics2D g2, String eventName, Rectangle2D eventBounds, int yLoc){
302     int eventStringHeight = (int) Math.ceil(eventBounds.getHeight());
303     int circleY = yLoc + (eventStringHeight+extraSpace*2)/2 - circleDiameter/2;
304     drawCircles(g2, circleY);
305     yLoc += eventStringHeight + extraSpace;
306     if (eventName.equals(errorEvent))
307         g2.setColor(ColorTheme.DARK_RED);
308     else
309         g2.setColor(ColorTheme.DARK_GREEN);
310
311     int x = bounds.x + circleDiameter/2 + 2*extraSpace + (int) Math.round((bounds.width-
312     circleDiameter-4*extraSpace-eventBounds.getWidth()) / 2.0);
313     Font orig = g2.getFont();
314     g2.setFont(eventFont);
315     g2.drawString(eventName, x, yLoc);
316     g2.setColor(Color.black);
317     yLoc += 2*extraSpace;
318     g2.drawLine(bounds.x, yLoc, bounds.x+bounds.width, yLoc);
319     g2.setColor(Color.BLACK);
320     g2.setFont(orig);
321     return yLoc + extraSpace;
322 }
323 private void drawBoundingRect(Graphics2D g2, boolean transparent) {
324     RoundRectangle2D rect = new RoundRectangle2D.Double(bounds.x, bounds.y, bounds.width,
325     bounds.height, 10, 10);
326     if (transparent) {
327         if (box instanceof StartBox) {
328             g2.setColor(ColorTheme.TRANS_GREEN);
329         } else if (box instanceof StopBox || box instanceof FailBox) {
330             g2.setColor(ColorTheme.TRANS_BLUE);
331         } else if (box instanceof InteractionBox) {
332             g2.setColor(ColorTheme.TRANS_RED);
333         } else {
334             g2.setColor(ColorTheme.TRANS_YELLOW);
335         }
336     } else {
337         if (box instanceof StartBox) {
338             g2.setColor(ColorTheme.NONTRANS_GREEN);
339         } else if (box instanceof StopBox || box instanceof FailBox) {
340             g2.setColor(ColorTheme.NONTRANS_BLUE);
341         } else if (box instanceof InteractionBox) {
342             g2.setColor(ColorTheme.NONTRANS_RED);
343         } else {
344             g2.setColor(ColorTheme.NONTRANS_YELLOW);
345         }
346 }
```

```
343     }
344     g2.fill(rect);
345     g2.setColor(Color.black);
346     g2.draw(rect);
347 }
348
349 private int drawTitle(Graphics2D g2){
350     String title = box.getTitle();
351     Rectangle2D titleBounds = titleLayout.getBounds();
352     int subTitleHeight = (int)Math.ceil(subTitleLayout.getBounds().getHeight());
353     int titleHeight = (int)Math.ceil(titleBounds.getHeight());
354     int sumHeight = titleHeight + subTitleHeight + titlesGap;
355     Font orig = g2.getFont();
356     g2.drawLine(bounds.x, bounds.y+sumHeight+titleSpace, bounds.x+bounds.width, bounds.y+
357     sumHeight+titleSpace);
358     g2.setFont(titleFont);
359     int x = bounds.x + (int)Math.round((bounds.width-titleBounds.getWidth()) / 2.0);
360     int y = bounds.y + titleHeight + titleSpace/2;
361     g2.drawString(title, x, y);
362     g2.setFont(subTitleFont);
363     x = bounds.x + (int)Math.round((bounds.width-subTitleLayout.getBounds().getWidth()) / 2.0)
364     ;
365     y = y+titlesGap+subTitleHeight;
366     g2.drawString(subTitle, x, y);
367     g2.setFont(orig);
368     return bounds.y+sumHeight+titleSpace;
369 }
370
371 public void markDeleted() {
372     TabbedPane p = box.getTabbedPane();
373     if (p != null)
374         p.removeTitleChangedListener(this);
375 }
376
377 public boolean isLeftCircle(int index) {
378     return index % 2 == 0;
379 }
380
381 public boolean isErrorCircle(int index) {
382     return events[index/2].equals(errorEvent);
383 }
384
385 private boolean isOverCircle(int circleIndex, Point point) {
386     Point rel = relativeCircleLocations[circleIndex];
387     Point p = new Point(rel.x+bounds.x, rel.y+bounds.y);
388     Rectangle r = new Rectangle(p.x-2, p.y-2, circleDiameter+4, circleDiameter+4);
389     if (r.contains(point))
390         return true;
391     return false;
392 }
393
394 public int getCircleIndex(Point point) {
395     for (int i = 0; i < relativeCircleLocations.length; i++) {
396         if (isOverCircle(i, point))
397             return i;
```

```
396         }
397         return -1;
398     }
399
400     public Point getCircleCenter(int index) {
401         Point rel = relativeCircleLocations[index];
402         return new Point(rel.x+bounds.x+circleDiameter/2, rel.y+bounds.y+circleDiameter/2);
403     }
404
405     public Dimension getSize() {
406         return new Dimension(bounds.width, bounds.height);
407     }
408
409     // index = -1 is allowed, it implies the circle focus is not painted.
410     public void setFocusCircle(int index) {
411         focusCircleIndex = index;
412     }
413
414     public boolean isOverControlPart(Point point) {
415         /*for (int i = 0; i < relativeCircleLocations.length; i++) {
416             if (isOverCircle(i, point))
417                 return false;
418         }*/
419         return bounds.contains(point);
420     }
421
422     @Override
423     public void titleChanged(String title) {
424         initialize(initialGraphics2D);
425     }
426
427     public List<RecordColumn> getInputs() {
428         return box.getInputs();
429     }
430
431     public Map<String,Type> getInputMap() {
432         List<RecordColumn> ins = getInputs();
433         Map<String,Type> ret = new HashMap<>();
434         for (RecordColumn i : ins) {
435             ret.put(i.getId(), i.getType());
436         }
437         return ret;
438     }
439
440     public List<RecordColumn> getOutputs(boolean includeError) {
441         List<RecordColumn> ret = box.getOutputs();
442         if (includeError)
443             ret.add(new RecordColumn(errorEvent, typeError));
444         return ret;
445     }
446
447     public Map<String,Type> getOutputMap(boolean includeError) {
448         List<RecordColumn> outs = getOutputs(includeError);
449         Map<String,Type> ret = new HashMap<>();
450         for (RecordColumn i : outs) {
```

```

451         ret.put(i.getId(), i.getType());
452     }
453     return ret;
454 }
455
456     public String getTitle() {
457         return box.getTitle();
458     }
459
460     public boolean isDataFlow() {
461         return box instanceof DataFlowBox;
462     }
463
464     public void resetCanFail() {
465         initialize(initialGraphics2D);
466     }
467
468     public ControlPartType getType() {
469         return box.getType();
470     }
471
472     public List<String> getEvents(boolean includeError) {
473         List<String> ret = new ArrayList<>(Arrays.asList(box.getEvents()));
474         if (includeError && box.canFail())
475             ret.add(errorEvent);
476         return ret;
477     }
478
479     public boolean canFail() {
480         return box.canFail();
481     }
482
483     public String getInteractionPlatform() {
484         String ret = box.getPlatform();
485         if (ret == null)
486             throw new RuntimeException("cannot get interaction platform from noninteraction box");
487         return ret;
488     }
489
490     public boolean isInsideOrIntersects(Rectangle r) {
491         return r.contains(bounds) || r.intersects(bounds);
492     }
493 }
```

LISTING E.114: gui/controlflow/ControlPart.java

```

1 package gui.controlflow;
2
3 public enum ControlPartType {
4     START, STOP, IF, DATAFLOW, INTERACTION, FAIL
5 }
```

LISTING E.115: gui/controlflow/ControlPartType.java

```
1 package gui.controlflow;
2
3 import lang.type.TypeConnection;
4
5 public class ControlPartTypeConnection extends TypeConnection {
6     private ControlPart outputPart;
7     private ControlPart inputPart;
8     private String outputName;
9     private String inputName;
10
11    public ControlPartTypeConnection(ControlPart outPart, String outId, ControlPart inPart, String
12        inId, boolean autoConnect) {
13        super(outPart.getOutputMap(true).get(outId), inPart.getInputMap().get(inId), autoConnect);
14        outputPart = outPart;
15        inputPart = inPart;
16        outputName = outId;
17        inputName = inId;
18        if (autoConnect)
19            autoConnect();
20    }
21
22    public ControlPart getOutputPart() {
23        return outputPart;
24    }
25
26    public ControlPart getInputPart() {
27        return inputPart;
28    }
29
30    public String getOutputName() {
31        return outputName;
32    }
33
34    public String getInputName() {
35        return inputName;
36    }
37
38    public void setInputName(String name) {
39        inputName = name;
40    }
41
42    public void setOutputName(String name) {
43        this.outputName = name;
44    }
45}
```

LISTING E.116: gui/controlflow/ControlPartTypeConnection.java

```
1 package gui;
2
3 import java.awt.BorderLayout;
4 import java.awt.Color;
5 import java.awt.Component;
6 import java.awt.ComponentOrientation;
```

```
7 import java.awt.Cursor;
8 import java.awt.Dimension;
9 import java.awt.FlowLayout;
10 import java.awt.Font;
11 import java.awt.GridBagConstraints;
12 import java.awt.GridBagLayout;
13 import java.awt.GridLayout;
14 import java.awt.Insets;
15 import java.awt.Point;
16 import java.awt.Rectangle;
17 import java.awt.event.ActionEvent;
18 import java.awt.event.ActionListener;
19 import java.awt.event.KeyAdapter;
20 import java.awt.event.KeyEvent;
21 import java.awt.event.MouseAdapter;
22 import java.awt.event.MouseEvent;
23 import java.awt.event.MouseListener;
24 import java.awt.event.MouseMotionListener;
25 import java.util.ArrayList;
26 import java.util.LinkedList;
27
28 import javax.swing.BorderFactory;
29 import javax.swing.BoxLayout;
30 import javax.swing.ComboBoxEditor;
31 import javax.swing.JButton;
32 import javax.swing.JComboBox;
33 import javax.swing.JComponent;
34 import javax.swing.JLabel;
35 import javax.swing.JList;
36 import javax.swing.JMenuItem;
37 import javax.swing.JPanel;
38 import javax.swing.JPopupMenu;
39 import javax.swing.JTextField;
40 import javax.swing.ListCellRenderer;
41 import javax.swing.border.Border;
42 import javax.swing.plaf.basic.BasicComboBoxUI;
43 import javax.swing.text.JTextComponent;
44
45 import util.UtilFont;
46 import util.UtilPrompt;
47 import lang.type.RecordColumn;
48 import lang.type.Type;
49 import lang.type.TypeAuto;
50 import lang.type.TypeBool;
51 import lang.type.TypeChangedListener;
52 import lang.type.TypeError;
53 import lang.type.TypeNumber;
54 import lang.type.TypeRecord;
55 import lang.type.TypeString;
56 import lang.type.TypeList;
57 import lang.type.TypeTime;
58 import lang.type.TypeUnknown;
59
60 @SuppressWarnings("serial")
61 public class TypeSelector extends JPanel {
```

```
62     final static int
63         TYPE_AUTO = 0,
64         TYPE_NUMBER = 1,
65         TYPE_STRING = 2,
66         TYPE_BOOL = 3,
67         TYPE_TIME = 4,
68         TYPE_TABLE = 5,
69         TYPE_ROW = 6,
70         TYPE_ERROR = 7;
71
72     private TypeSelectorBox selectorBox;
73     private ArrayList<TypeChangedListener> typeChangedListeners = new ArrayList<>();
74
75     public TypeSelector(String title) {
76         this(title, false);
77     }
78
79     public TypeSelector(String title, boolean allowAuto) {
80         setLayout(new FlowLayout(FlowLayout.LEADING, 10, 10));
81         selectorBox = new TypeSelectorBox(this, allowAuto);
82         JPanel outer = new JPanel();
83         outer.setBorder(BorderFactory.createTitledBorder(title));
84         JPanel inner = new JPanel(new BorderLayout());
85         inner.add(selectorBox);
86         inner.setBackground(Color.white);
87         inner.setBorder(BorderFactory.createLoweredBevelBorder());
88         outer.add(inner);
89         add(outer);
90     }
91
92     public Type getType() {
93         return selectorBox.getType();
94     }
95
96     public void addTypeChangedListener(TypeChangedListener listener) {
97         typeChangedListeners.add(listener);
98     }
99
100    void typeChanged(Type t) {
101        for (TypeChangedListener list : typeChangedListeners)
102            list.typeChanged(t);
103    }
104 }
105
106 @SuppressWarnings("serial")
107 class TypeSelectorBox extends JComboBox<Integer> {
108     TypeEditor editor;
109     private TypeSelector typeSelector;
110     private Type prevType;
111
112     TypeSelectorBox(TypeSelector selector, boolean allowAuto) {
113         this(null, selector, allowAuto);
114     }
115
116     TypeSelectorBox(TypeEditor parent, boolean allowAuto) {
```

```
117     this(parent, null, allowAuto);
118 }
119
120 private static Integer[] getTypeList(boolean allowAuto) {
121     if (!allowAuto)
122         return new Integer[] {
123             TypeSelector.TYPE_NUMBER,
124             TypeSelector.TYPE_STRING,
125             TypeSelector.TYPE_BOOL,
126             //TypeSelector.TYPE_TIME,
127             TypeSelector.TYPE_ROW,
128             TypeSelector.TYPE_TABLE,
129             TypeSelector.TYPE_ERROR
130         };
131     return new Integer[] {
132         TypeSelector.TYPE_AUTO,
133         TypeSelector.TYPE_NUMBER,
134         TypeSelector.TYPE_STRING,
135         TypeSelector.TYPE_BOOL,
136         //TypeSelector.TYPE_TIME,
137         TypeSelector.TYPE_ROW,
138         TypeSelector.TYPE_TABLE,
139         TypeSelector.TYPE_ERROR
140     };
141 }
142
143 TypeSelectorBox(TypeEditor parent, TypeSelector selector, boolean allowAuto) {
144     super(getTypeList(allowAuto));
145     typeSelector = selector;
146
147     applyComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);
148
149     setEditable(true);
150     setRenderer(new TypeSelectorRenderer());
151
152     editor = new TypeEditor(parent, this, null, allowAuto);
153     setEditor(editor);
154     setUI(new TypeSelectorUI(editor.getPanel()));
155     if (allowAuto)
156         setSelectedIndex(TypeSelector.TYPE_AUTO);
157     else
158         setSelectedItem(null);
159 }
160
161 public Type getType() {
162     if (editor == null)
163         return null;
164     return editor.getType();
165 }
166
167 @Override
168 public boolean isValidRoot() {
169     return false;
170 }
171
```

```
172     @Override
173     public void addMouseListener(MouseListener t) {}
174
175     void typeChanged() {
176         if (typeSelector != null) {
177             Type t = getType();
178             if (t != null && (prevType == null || !prevType.equals(t))) {
179                 prevType = t;
180                 typeSelector.typeChanged(t);
181             }
182         }
183     }
184 }
185
186 @SuppressWarnings("serial")
187 class TypeEditorPanel extends JPanel implements MouseListener {
188     private boolean mouseIsOverText = false;
189
190     TypeEditor.TypeJPanel typeJPanel;
191
192     TypeEditorPanel() {
193         addMouseListener(this);
194     }
195
196     public void addTypeJPanel(TypeEditor.TypeJPanel p) {
197         if (typeJPanel != null) {
198             throw new RuntimeException("forgot to remove TypeJPanel?");
199         }
200         typeJPanel = p;
201         super.add(p);
202     }
203
204     @Override
205     public void remove(Component c) {
206         typeJPanel = null;
207         super.remove(c);
208     }
209
210     public Type getType() {
211         return typeJPanel.getType();
212     }
213
214     @Override
215     public Component add(Component c) {
216         throw new RuntimeException("use addTypeJPanel instead");
217     }
218
219     @Override
220     public void mouseClicked(MouseEvent e) {
221         Component c = this.getComponentAt(e.getPoint());
222         if (c == null || c == this)
223             return;
224         for (MouseListener m : c.getMouseListeners())
225             m.mouseClicked(e);
226     }
```

```
227
228     private enum MouseEventEventType {EVENT_PRESS, EVENT_RELEASE, EVENT_CLICK,
229         EVENT_ENTERED, EVENT_EXITED, EVENT_MOVED, EVENT_DRAGGED};
230
231     private void handleMouseEventType(Component c, MouseEvent e, MouseEventEventType t) {
232         Point first = e.getPoint();
233         if (!c.contains(first))
234             return;
235         if (c instanceof JTextComponent)
236             mouseIsOverText = true;
237
238         for (MouseListener m : c.getMouseListeners()) {
239             switch (t) {
240                 case EVENT_PRESS:
241                     m.mousePressed(e);
242                     break;
243                 case EVENT_RELEASE:
244                     m.mouseReleased(e);
245                     break;
246                 case EVENT_CLICK:
247                     m.mouseClicked(e);
248                     break;
249                 case EVENT_ENTERED:
250                     m.mouseEntered(e);
251                     break;
252                 case EVENT_EXITED:
253                     m.mouseExited(e);
254                     break;
255                 default:
256                     break;
257             }
258         }
259         for (MouseMotionListener m : c.getMouseMotionListeners()) {
260             switch (t) {
261                 case EVENT_DRAGGED:
262                     m.mouseDragged(e);
263                     break;
264                 case EVENT_MOVED:
265                     m.mouseMoved(e);
266                     break;
267                 default:
268                     break;
269             }
270         }
271
272         Component c2 = c.getComponentAt(first);
273         if (c2 == null || c2 == c)
274             return;
275         Point a = c.getLocationOnScreen();
276         Point b = c2.getLocationOnScreen();
277         int x = first.x + a.x-b.x;
278         int y = first.y + a.y-b.y;
279         MouseEvent next = new MouseEvent(c2, e.getID(), e.getWhen(), e.getModifiers(),
280             x, y, e.getClickCount(), e.isPopupTrigger());
281         handleMouseEventType(c2, next, t);
```

```
282     }
283
284     boolean isMouseOverText() {
285         return mouseIsOverText;
286     }
287
288     private void handleMouseEvent(MouseEvent e, MouseEvent.Type t) {
289         mouseIsOverText = false;
290         Component c = this.getComponentAt(e.getPoint().x-TypeSelectorUI.minimumSize.width, e.
291             getPoint().y);
292         if (c == null || c == this)
293             return;
294         e.setSource(c);
295         MouseEvent real = new MouseEvent(c, e.getID(), e.getWhen(), e.getModifiers(),
296             e.getX()-TypeSelectorUI.minimumSize.width, e.getY(),e.getClickCount(), e.
297             isPopupTrigger());
298         handleMouseEventType(c, real, t);
299     }
300
301     @Override
302     public void mousePressed(MouseEvent e) {
303         handleMouseEvent(e, MouseEvent.Type.EVENT_PRESS);
304     }
305
306     @Override
307     public void mouseReleased(MouseEvent e) {
308         handleMouseEvent(e, MouseEvent.Type.EVENT_RELEASE);
309     }
310
311     @Override
312     public void mouseEntered(MouseEvent e) {
313         handleMouseEvent(e, MouseEvent.Type.EVENT_ENTERED);
314     }
315
316     @Override
317     public void mouseExited(MouseEvent e) {
318         handleMouseEvent(e, MouseEvent.Type.EVENT_EXITED);
319     }
320
321     public void mouseMoved(MouseEvent e) {
322         handleMouseEvent(e, MouseEvent.Type.EVENT_MOVED);
323     }
324
325     public void mouseDragged(MouseEvent e) {
326         handleMouseEvent(e, MouseEvent.Type.EVENT_DRAGGED);
327     }
328
329     class TypeSelectorUI extends BasicComboBoxUI {
330         final static Dimension minimumSize = new Dimension(22, 22);
331         final static Font typeFont = UtilFont.boldTextFont;
332         final ArrayList<MouseListener> listeners = new ArrayList<>();
333         final TypeEditorPanel rightSide;
334
335         TypeSelectorUI(TypeEditorPanel right) {
```

```
335         this.rightSide = right;
336     }
337
338     @Override
339     protected JButton createArrowButton() {
340         JButton ret = new TypeButton();
341         return ret;
342     }
343
344     @SuppressWarnings("serial")
345     private class TypeButton extends JButton implements MouseListener, MouseMotionListener {
346         TypeButton() {
347             super.addMouseListener(this);
348             super.addMouseMotionListener(this);
349             setMargin(new Insets(0, 0, 0, 0));
350             setFocusPainted(false);
351            setFont(new Font(UtilFont.fontName, Font.PLAIN, 13));
352             setText("\u21d5");
353         }
354         @Override
355         public int getWidth() {
356             return TypeSelectorUI.minimumSize.width;
357         }
358
359         @Override
360         public void addMouseListener(MouseListener listener) {
361             listeners.add(listener);
362         }
363
364         @Override
365         public void mouseClicked(MouseEvent e) {
366             if (e.getPoint().x <= this.getLocation().x+getWidth()) {
367                 for (MouseListener t : listeners) {
368                     t.mouseClicked(e);
369                 }
370             } else {
371                 rightSide.mouseClicked(e);
372             }
373         }
374         @Override
375         public void mousePressed(MouseEvent e) {
376             if (e.getPoint().x <= this.getLocation().x+getWidth()) {
377                 for (MouseListener t : listeners) {
378                     t.mousePressed(e);
379                 }
380             } else {
381                 rightSide.mousePressed(e);
382             }
383         }
384
385         @Override
386         public void mouseReleased(MouseEvent e) {
387             if (e.getPoint().x <= this.getLocation().x+getWidth()) {
388                 for (MouseListener t : listeners) {
389                     t.mouseReleased(e);
```

```
390         }
391     } else {
392         rightSide.mouseReleased(e);
393     }
394 }
395
396 @Override
397 public void mouseEntered(MouseEvent e) {
398     if (e.getPoint().x <= this.getLocation().x+getWidth()) {
399         //for (MouseListener t : listeners) {
400             //t.mouseEntered(e);
401         //}
402         setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
403     }
404 }
405
406 @Override
407 public void mouseExited(MouseEvent e) {
408     //for (MouseListener t : listeners) {
409         //t.mouseExited(e);
410     //}
411     setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
412     rightSide.mouseExited(e);
413 }
414
415 @Override
416 public void mouseDragged(MouseEvent e) {
417     Rectangle rightBounds = new Rectangle(getBounds());
418     rightBounds.x += getWidth();
419     rightBounds.width -= getWidth();
420     if (rightBounds.contains(e.getPoint())) {
421         rightSide.mouseDragged(e);
422         if (rightSide.isMouseOverText()) {
423             //setCursor(new Cursor(Cursor.TEXT_CURSOR));
424             setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
425         } else {
426             setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
427         }
428     } else {
429         setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
430     }
431 }
432
433 @Override
434 public void mouseMoved(MouseEvent e) {
435     Rectangle rightBounds = new Rectangle(getBounds());
436     rightBounds.x += getWidth();
437     rightBounds.width -= getWidth();
438     if (rightBounds.contains(e.getPoint())) {
439         rightSide.mouseMoved(e);
440         if (rightSide.isMouseOverText()) {
441             setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
442         } else {
443             setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
444         }
445 }
```

```
445         } else {
446             setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
447         }
448     }
449 }
450 }
451
452 class TypeEditor implements ComboBoxEditor {
453     private final static Dimension minDimension = new Dimension(80,0);
454     private TypeJPanel currentComponent;
455     private TypeEditorPanel panel = new TypeEditorPanel();
456
457     private Integer currentItem;
458     private TypeSelectorBox selectorBox;
459     private TypeEditor parent;
460     private boolean allowAuto;
461
462     public TypeEditor(TypeEditor parentEditor, TypeSelectorBox selectorBox,
463                     Integer initialValue, boolean allowAuto) {
464         this.parent = parentEditor;
465         this.selectorBox = selectorBox;
466         this.panel.setLayout(new GridLayout());
467         setCurrentItemAndComponent(initialValue);
468         this.allowAuto = allowAuto;
469     }
470
471     Type getType() {
472         return panel.getType();
473     }
474
475     TypeEditorPanel getPanel() {
476         return panel;
477     }
478
479     TypeSelectorBox getTypeSelectorBox() {
480         return selectorBox;
481     }
482
483     TypeEditor getParent() {
484         return parent;
485     }
486
487     void setCurrentItemAndComponent(Integer idx) {
488         currentItem = idx;
489         TypeJPanel newComponent;
490         if (idx == null) {
491             newComponent = getNullComponent();
492         } else {
493             switch (idx) {
494                 case TypeSelector.TYPE_NUMBER:
495                     newComponent = getNumberComponent();
496                     break;
497                 case TypeSelector.TYPE_STRING:
498                     newComponent = getStringComponent();
499                     break;
```

```
500         case TypeSelector.TYPE_BOOL:
501             newComponent = getBoolComponent();
502             break;
503         case TypeSelector.TYPE_TIME:
504             newComponent = getTimeComponent();
505             break;
506         case TypeSelector.TYPE_AUTO:
507             newComponent = getAutoComponent();
508             break;
509         case TypeSelector.TYPE_TABLE:
510             newComponent = getTableComponent();
511             break;
512         case TypeSelector.TYPE_ROW:
513             newComponent = getRowComponent();
514             break;
515         case TypeSelector.TYPE_ERROR:
516             newComponent = getErrorComponent();
517             break;
518         default:
519             throw new IllegalArgumentException("illegal type index " + idx);
520         }
521     }
522     revalidate(newComponent);
523 }
524
525 private int getRowJPanelHeight(RowJPanel rowPanel, int nest) {
526     assert nest >= 1;
527
528     int maxHeight = 0;
529     for (ColumnJPanel p : rowPanel.columns) {
530         TypeSelectorBox tb = p.selectorBox;
531         TypeJPanel typePanel = tb.editor.panel.typeJPanel;
532         if (nest > 1) {
533             while (typePanel instanceof TableJPanel) {
534                 TableJPanel table = (TableJPanel)typePanel;
535                 typePanel = table.selector.editor.currentComponent;
536             }
537             if (typePanel instanceof RowJPanel) {
538                 RowJPanel child = (RowJPanel)typePanel;
539                 int h = getRowJPanelHeight(child, nest-1);
540                 maxHeight = Math.max(maxHeight, h);
541             }
542         } else {
543             Dimension dim = tb.getSize();
544             maxHeight = Math.max(dim.height, maxHeight);
545         }
546     }
547     return maxHeight;
548 }
549
550 private void revalidate(TypeJPanel newComponent) {
551     Dimension s = newComponent.getPreferredSize();
552     if (s.height < TypeSelectorUI.minimumSize.height)
553         s.height = TypeSelectorUI.minimumSize.height;
554     newComponent.setPreferredSize(s);
```

```
555
556     if (currentComponent != null)
557         panel.remove(currentComponent);
558     panel.addTypeJPanel(newComponent);
559     currentComponent = newComponent;
560
561     Dimension d1 = panel.getPreferredSize();
562     Dimension d2 = selectorBox.getSize();
563
564     TypeEditor edi = parent;
565     RowJPanel rowPanel = null;
566     int nest = 0;
567     while (edi != null) {
568         if (edi.currentComponent instanceof RowJPanel) {
569             rowPanel = (RowJPanel)edi.currentComponent;
570             nest += 1;
571         } else if (!(edi.currentComponent instanceof TableJPanel)) {
572             break;
573         }
574         edi = edi.parent;
575     }
576     if (rowPanel != null) {
577         d1.height = Math.max(d1.height, getRowJPanelHeight(rowPanel, nest));
578     }
579     if (newComponent instanceof RowJPanel) {
580         updateSize(d1.width-d2.width+30, d1.height-d2.height);
581     } else {
582         updateSize(d1.width-d2.width+30, d1.height-d2.height);
583     }
584 }
585
586     private void updateSize(int width, int height) {
587         Dimension d = selectorBox.getSize();
588         d.width += width;
589         d.height += height;
590         selectorBox.setPreferredSize(d);
591         if (parent != null) {
592             parent.updateSize(width, height);
593         } else {
594             selectorBox.revalidate();
595             selectorBox.typeChanged();
596         }
597     }
598
599     @SuppressWarnings("serial")
600     abstract class TypeJPanel extends JPanel {
601         TypeJPanel() {
602             super(new BorderLayout());
603             setBackground(Color.white);
604         }
605         abstract Type getType();
606     }
607
608     @SuppressWarnings("serial")
609     class ColumnJPanel extends JPanel {
```

```
610     JTextField text;
611     TypeSelectorBox selectorBox;
612
613     ColumnJPanel() {
614         setBackground(Color.white);
615     }
616
617     void setText(JTextField text) {
618         this.text = text;
619         super.add(text);
620     }
621     void setTypeSelectorBox(TypeSelectorBox b) {
622         selectorBox = b;
623         super.add(b);
624     }
625     String getText() {
626         return text.getText();
627     }
628     Type getType() {
629         return selectorBox.getType();
630     }
631     @Override
632     public Component add(Component c) {
633         throw new RuntimeException("use setText/setTypeSelectorBox");
634     }
635 }
636
637 @SuppressWarnings("serial")
638 class RowJPanel extends TypeJPanel {
639     ArrayList<ColumnJPanel> columns = new ArrayList<>();
640     RowJPanel() {
641         setLayout(new BoxLayout(this, BoxLayout.X_AXIS));
642     }
643
644     public void addColumn(ColumnJPanel c) {
645         columns.add(c);
646         add(c);
647     }
648
649     @Override
650     Type getType() {
651         ArrayList<RecordColumn> rowcols = new ArrayList<>();
652         for (ColumnJPanel c : columns) {
653             rowcols.add(new RecordColumn(c.getText(), c.getType()));
654         }
655         try {
656             return new TypeRecord(rowcols);
657         } catch (IllegalArgumentException e) {
658             return new TypeUnknown(e.getMessage());
659         }
660     }
661 }
662
663 class AddColumnListener implements ActionListener {
664     private RowJPanel[] row;
```

```
665     private JPanel column;
666     private boolean insertAfter;
667
668     AddColumnListener(RowJPanel[] row, JPanel column, boolean insertAfter) {
669         this.row = row;
670         this.column = column;
671         this.insertAfter = insertAfter;
672     }
673
674     @Override
675     public void actionPerformed(ActionEvent e) {
676         LinkedList<ColumnJPanel> after = new LinkedList<>();
677         ArrayList<ColumnJPanel> all = row[0].columns;
678
679         int sourceIndex = all.size()-1;
680         for (; sourceIndex >= 0; sourceIndex--) {
681             if (all.get(sourceIndex) == column)
682                 break;
683             after.add(0, all.get(sourceIndex));
684         }
685
686         if (sourceIndex < 0)
687             throw new RuntimeException("unable to find right click source component " + column
688 );
689         if (insertAfter)
690             sourceIndex += 1;
691         else
692             after.add(0, all.get(sourceIndex));
693
694         while (sourceIndex < all.size()) {
695             all.remove(sourceIndex);
696         }
697
698         RowJPanel nrow = addColumnToRow(row);
699         for (ColumnJPanel c : after)
700             nrow.addColumn(c);
701         revalidate(nrow);
702     }
703 }
704
705 class DeleteColumnListener implements ActionListener {
706     private RowJPanel[] row;
707     private JPanel column;
708
709     DeleteColumnListener(RowJPanel[] row, JPanel column) {
710         this.row = row;
711         this.column = column;
712     }
713
714     @Override
715     public void actionPerformed(ActionEvent e) {
716         ArrayList<ColumnJPanel> all = row[0].columns;
717         if (all.size() == 1) {
718             selectorBox.setSelectedItem(null);
```

```
719         return;
720     }
721     RowJPanel nrow = new RowJPanel();
722     for (ColumnJPanel p : all) {
723         if (p == column)
724             continue;
725         nrow.addColumn(p);
726     }
727     row[0] = nrow;
728     revalidate(nrow);
729 }
730 }
731
732 private RowJPanel addColumnToRow(RowJPanel[] rowArr) {
733     RowJPanel row = rowArr[0];
734
735     RowJPanel nrow = new RowJPanel();
736
737     for (ColumnJPanel c : row.columns) {
738         nrow.addColumn(c);
739     }
740
741     ColumnJPanel column = new ColumnJPanel();
742     GridBagLayout columnLayout = new GridBagLayout();
743     column.setLayout(columnLayout);
744     GridBagConstraints constraints = new GridBagConstraints();
745
746     JTextField text = new JTextField();
747     text.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
748     text.setBorder(createRecordBorder());
749     text.addKeyListener(new KeyAdapter() {
750         @Override
751         public void keyReleased(KeyEvent e) {
752             TypeEditor editor = TypeEditor.this;
753             while (editor.getParent() != null) {
754                 editor = editor.getParent();
755             }
756             editor.getTypeSelectorBox().typeChanged();
757         }
758     });
759     UtilPrompt.setPrompt("Column name", text);
760
761     Font font = UtilFont.textFont;
762     text.setFont(font);
763
764     TypeSelectorBox s = new TypeSelectorBox(this, allowAuto);
765
766     constraints.fill = GridBagConstraints.HORIZONTAL;
767     constraints.weightx = 0;
768     constraints.weighty = 0;
769     constraints.gridx = 0;
770     constraints.gridy = 0;
771     constraints.gridwidth = 1;
772     constraints.gridheight = 1;
773     columnLayout.setConstraints(text, constraints);
```

```
774     column.setText(text);
775     text.setPreferredSize(text.getPreferredSize());
776     constraints.fill = GridBagConstraints.BOTH;
777     constraints.weightx = 0;
778     constraints.weighty = 100;
779     constraints.gridx = 0;
780     constraints.gridy = 1;
781     constraints.gridwidth = 1;
782     constraints.gridheight = 1;
783     columnLayout.setConstraints(s, constraints);
784     column.setTypeSelectorBox(s);
785
786     nrow.addColumn(column);
787     rowArr[0] = nrow;
788
789     ActionListener a1 = new AddColumnListener(rowArr, column, true);
790     JMenuItem menuItem1 = new JMenuItem("Insert column after");
791     menuItem1.addActionListener(a1);
792
793     ActionListener a2 = new AddColumnListener(rowArr, column, false);
794     JMenuItem menuItem2 = new JMenuItem("Insert column before");
795     menuItem2.addActionListener(a2);
796
797     ActionListener a3 = new DeleteColumnListener(rowArr, column);
798     JMenuItem menuItem3 = new JMenuItem("Delete column");
799     menuItem3.addActionListener(a3);
800
801     JPopupMenu popup = new JPopupMenu();
802     popup.add(menuItem1);
803     popup.add(menuItem2);
804     popup.add(menuItem3);
805     text.addMouseListener(new MouseAdapter() {
806         public void mousePressed(MouseEvent e) {
807             maybeShowPopup(e);
808         }
809         public void mouseReleased(MouseEvent e) {
810             maybeShowPopup(e);
811         }
812         private void maybeShowPopup(MouseEvent e) {
813             if (e.isPopupTrigger()) {
814                 popup.show(e.getComponent(), e.getX(), e.getY());
815             }
816         }
817     });
818
819     return nrow;
820 }
821
822 private TypeJPanel getRowComponent() {
823     RowJPanel[] row = {new RowJPanel()};
824     return addColumnToRow(row);
825 }
826
827 @SuppressWarnings("serial")
828 class TableJPanel extends TypeJPanel {
```

```
829     TypeSelectorBox selector;
830     TableJPanel(TypeSelectorBox b) {
831         selector = b;
832     }
833     public Type getType() {
834         return new TypeList(selector.getType());
835     }
836 }
837
838 private TypeJPanel getTableComponent() {
839     JLabel text = new JLabel("List");
840     text.setFont(TypeSelectorUI.typeFont);
841     text.setPreferredSize(new Dimension(30,0));
842     TypeSelectorBox s = new TypeSelectorBox(this, allowAuto);
843
844     TableJPanel ret = new TableJPanel(s);
845     ret.setLayout(new BorderLayout());
846     ret.add(text, BorderLayout.LINE_START);
847     ret.add(s, BorderLayout.LINE_END);
848     return ret;
849 }
850
851 @SuppressWarnings("serial")
852 class AutoJPanel extends TypeJPanel {
853     public Type getType() {
854         return new TypeAuto();
855     }
856 }
857 private TypeJPanel getAutoComponent() {
858     JLabel f = new JLabel("Auto");
859     f.setFont(TypeSelectorUI.typeFont);
860     f.setPreferredSize(minDimension);
861     AutoJPanel p = new AutoJPanel();
862     p.add(f);
863     return p;
864 }
865
866 @SuppressWarnings("serial")
867 class NullJPanel extends TypeJPanel {
868     public Type getType() {
869         return new TypeUnknown("Missing type");
870     }
871 }
872
873 private TypeJPanel getNullComponent() {
874     JLabel f = new JLabel("select type");
875     Font font = UtilFont.errorTextFont;
876     f.setFont(font);
877     f.setForeground(Color.red);
878     f.setPreferredSize(minDimension);
879     NullJPanel p = new NullJPanel();
880     p.add(f);
881     return p;
882 }
883 @SuppressWarnings("serial")
```

```
884     class ErrorJPanel extends TypeJPanel {
885         public Type getType() {
886             return new TypeError();
887         }
888     }
889     private TypeJPanel getErrorComponent() {
890         JLabel f = new JLabel("Error");
891         f.setFont(TypeSelectorUI.typeFont);
892         f.setPreferredSize(minDimension);
893         ErrorJPanel p = new ErrorJPanel();
894         p.add(f);
895         return p;
896     }
897     @SuppressWarnings("serial")
898     class NumberJPanel extends TypeJPanel {
899         public Type getType() {
900             return new TypeNumber();
901         }
902     }
903     private TypeJPanel getNumberComponent() {
904         JLabel f = new JLabel("Number");
905         f.setFont(TypeSelectorUI.typeFont);
906         f.setPreferredSize(minDimension);
907         NumberJPanel p = new NumberJPanel();
908         p.add(f);
909         return p;
910     }
911     @SuppressWarnings("serial")
912     class StringJPanel extends TypeJPanel {
913         public Type getType() {
914             return new TypeString();
915         }
916     }
917     private TypeJPanel getStringComponent() {
918         JLabel f = new JLabel("String");
919         f.setFont(TypeSelectorUI.typeFont);
920         f.setPreferredSize(minDimension);
921         StringJPanel p = new StringJPanel();
922         p.add(f);
923         return p;
924     }
925     @SuppressWarnings("serial")
926     class TimeJPanel extends TypeJPanel {
927         public Type getType() {
928             return new TypeTime();
929         }
930     }
931     private TypeJPanel getTimeComponent() {
932         JLabel f = new JLabel("Time");
933         f.setFont(TypeSelectorUI.typeFont);
934         f.setPreferredSize(minDimension);
935         TimeJPanel p = new TimeJPanel();
936         p.add(f);
937         return p;
938     }
```

```
939     @SuppressWarnings("serial")
940     class BoolJPanel extends TypeJPanel {
941         public Type getType() {
942             return new TypeBool();
943         }
944     }
945     private TypeJPanel getBoolComponent() {
946         JLabel f = new JLabel("Boolean");
947         f.setFont(TypeSelectorUI.typeFont);
948         f.setPreferredSize(minDimension);
949         BoolJPanel p = new BoolJPanel();
950         p.add(f);
951         return p;
952     }
953
954     @Override
955     public void addActionListener(ActionListener l) {}
956
957     @Override
958     public Component getEditorComponent() {
959         return panel;
960     }
961
962     @Override
963     public Object getItem() {
964         return currentItem;
965     }
966
967     @Override
968     public void removeActionListener(ActionListener l) {}
969
970     @Override
971     public void selectAll() {}
972
973     @Override
974     public void setItem(Object newValue) {
975         setCurrentItemAndComponent((Integer)newValue);
976     }
977
978     public static Border createRecordBorder() {
979         return BorderFactory.createEtchedBorder();
980     }
981 }
982
983 class TypeSelectorRenderer implements ListCellRenderer<Integer> {
984
985     public Component getListCellRendererComponent(JList<? extends Integer> list, Integer value,
986         int index, boolean isSelected, boolean cellHasFocus) {
987
988         JComponent ret = getType(value);
989
990         if (isSelected) {
991             ret.setBackground(list.getSelectionBackground());
992             ret.setForeground(list.getSelectionForeground());
993         } else {
```

```
994         ret.setBackground(Color.white);
995         ret.setForeground(Color.black);
996     }
997
998     return ret;
999 }
1000
1001     private JComponent getType(Integer idx) {
1002         JComponent ret;
1003         switch (idx) {
1004             case TypeSelector.TYPE_NUMBER:
1005                 ret = getLeafType("Number");
1006                 break;
1007             case TypeSelector.TYPE_STRING:
1008                 ret = getLeafType("String");
1009                 break;
1010             case TypeSelector.TYPE_BOOL:
1011                 ret = getLeafType("Boolean");
1012                 break;
1013             case TypeSelector.TYPE_TIME:
1014                 ret = getLeafType("Time");
1015                 break;
1016             case TypeSelector.TYPE_TABLE:
1017                 ret = getLeafType("List");
1018                 break;
1019             case TypeSelector.TYPE_ROW:
1020                 ret = getLeafType("Record");
1021                 break;
1022             case TypeSelector.TYPE_ERROR:
1023                 ret = getLeafType("Error");
1024                 break;
1025             case TypeSelector.TYPE_AUTO:
1026                 ret = getLeafType("Auto");
1027                 break;
1028             default:
1029                 throw new IllegalArgumentException("unexpected type selector index " + idx);
1030         }
1031         return ret;
1032     }
1033
1034     private JComponent getLeafType(String type) {
1035         JPanel ret = new JPanel(new GridLayout(1,1));
1036         ret.add(getLeafTextField(type));
1037         return ret;
1038     }
1039
1040     private JLabel getLeafTextField(String type) {
1041         JLabel lbl1 = new JLabel(type);
1042         lbl1.setFont(TypeSelectorUI.typeFont);
1043         lbl1.setBorder(BorderFactory.createLoweredBevelBorder());
1044         return lbl1;
1045     }
1046 }
```

LISTING E.117: gui/TypeSelector.java

```
1 package gui.temp;
2
3 import javax.swing.JPanel;
4
5 /*
6 import gui.DataFlowPanel;
7 import gui.TypeSelector;
8 import gui.dataflow.DataPart;
9 import gui.dataflow.Port;
10
11 import java.awt.Color;
12 import java.awt.Component;
13 import java.awt.Container;
14 import java.awt.Dimension;
15 import java.awt.FlowLayout;
16 import java.awt.Graphics2D;
17 import java.awt.GridBagConstraints;
18 import java.awt.GridBagLayout;
19 import java.awt.Point;
20 import java.awt.event.ActionEvent;
21 import java.awt.event.ActionListener;
22 import java.awt.event.ItemEvent;
23 import java.awt.event.ItemListener;
24 import java.awt.event.KeyEvent;
25 import java.awt.event.KeyListener;
26 import java.awt.event.MouseEvent;
27 import java.awt.event.MouseListener;
28 import java.awt.event.WindowAdapter;
29 import java.awt.event.WindowEvent;
30 import java.util.ArrayList;
31
32 import javax.swing.BorderFactory;
33 import javax.swing.JButton;
34 import javax.swing.JComboBox;
35 import javax.swing.JDialog;
36 import javax.swing.JLabel;
37 import javax.swing.JOptionPane;
38 import javax.swing.JPanel;
39 import javax.swing.JTable;
40 import javax.swing.JTextField;
41 import javax.swing.event.TableModelEvent;
42 import javax.swing.event.TableModelListener;
43 import javax.swing.table.DefaultTableModel;
44 import javax.swing.table.TableCellRenderer;
45 import javax.swing.table.TableModel;
46
47 import util.UtilScrollPane;
48 import lang.type.Type;
49 */
50 @SuppressWarnings("serial")
51 public class DataPartBuilder extends JPanel{
52 }
53 /*
54     private static final long serialVersionUID = 1L;
```

```
55     private JLabel inputLabel;
56     private JLabel outputLabel;
57     private JComboBox<String> inputComboBox;
58     private JComboBox<String> outputComboBox;
59     private JButton okButton;
60     private JTable tableInput;
61     private JTable tableOutput;
62     private int rowNumber = 0;
63     private int rowNumber1 = 0;
64     private String[] numbers = {"0", "1", "2", "3", "4", "5", "6"};
65     private DrawPanel drawPanel;
66     private JPanel topPanel;
67     private DataPart dataPart;
68     private DataFlowPanel window;
69     private JTextField componentTitle = new JTextField("Component Title");
70
71     private GridBagLayout topLayout = new GridBagLayout();
72     private GridBagLayout mainLayout = new GridBagLayout();
73     private GridBagConstraints constraints = new GridBagConstraints();
74
75     public static Port[] getTablePorts(JTable t) {
76         return getTableModelPorts(t.getModel());
77     }
78
79     public static Port[] getTableModelPorts(TableModel m1) {
80         TModel m = (TModel)m1;
81         Port[] ret = new Port[m.getRowCount()];
82         for (int i = 0; i < ret.length; i++) {
83             ret[i] = new Port(m.getValueAt(i, 0).toString(), m.getTypeAt(i));
84         }
85         return ret;
86     }
87
88     public DataPartBuilder(DataFlowPanel window) {
89         setLayout(mainLayout);
90         topPanel = new JPanel(topLayout);
91
92         this.window = window;
93
94         componentTitle.addKeyListener(new KeyListener() {
95
96             @Override
97             public void keyTyped(KeyEvent e) {
98                 drawPanel.repaint();
99
100            }
101
102            @Override
103            public void keyReleased(KeyEvent e) {}
104
105            @Override
106            public void keyPressed(KeyEvent e) {}
107        });
108        //label for input
109        inputLabel = new JLabel("Select the Number of Inputs");
```

```

110     addComponent(topLayout, topPanel, inputLabel, 0, 0, 1, 1);
111
112     //label for output
113     outputLabel = new JLabel("Select the Number of Outputs");
114     addComponent(topLayout, topPanel, outputLabel, 0, 1, 1, 1);
115
116     //drop down box for input
117     inputComboBox = new JComboBox<String>(numbers);
118     inputComboBox.setMaximumRowCount(5);
119     inputComboBox.addItemListener(new ItemListener() {
120         @Override
121         public void itemStateChanged(ItemEvent e) {
122             if(e.getStateChange() == ItemEvent.SELECTED){
123                 int row = inputComboBox.getSelectedIndex();
124                 DefaultTableModel model = (DefaultTableModel) tableInput.getModel();
125                 for(int x=rowNumber-1; x>=0;x--){
126                     model.removeRow(x);
127                 }
128                 for(int i=0; i<row; i++){
129                     model.addRow(new String[]{"Input"+(i+1), "Select"});
130                 }
131                 rowNumber = row;
132                 drawPanel.repaint();
133             }
134         }
135     });
136     constraints.fill = GridBagConstraints.HORIZONTAL;
137     constraints.weightx = 1;
138     constraints.weighty = 0;
139     addComponent(topLayout, topPanel, inputComboBox, 1, 0, 1, 1);
140
141     //drop down box for output
142     outputComboBox = new JComboBox<String>(numbers);
143     outputComboBox.setMaximumRowCount(5);
144     outputComboBox.addItemListener(new ItemListener() {
145         @Override
146         public void itemStateChanged(ItemEvent e) {
147             if(e.getStateChange() == ItemEvent.SELECTED){
148                 int row = outputComboBox.getSelectedIndex();
149                 DefaultTableModel model = (DefaultTableModel) tableOutput.getModel();
150                 for(int x=rowNumber1-1;x>=0; x--){
151                     model.removeRow(x);
152                 }
153                 for(int i=0; i<row; i++){
154                     model.addRow(new String[]{"Output"+(i+1), "Select"});
155                 }
156                 rowNumber1 = row;
157                 drawPanel.repaint();
158             }
159         }
160     });
161     constraints.fill = GridBagConstraints.HORIZONTAL;
162     constraints.weightx = 1;
163     constraints.weighty = 0;
164     addComponent(topLayout, topPanel, outputComboBox, 1, 1, 1, 1);

```

```

165
166     //table for input
167     tableInput = new TTable("Input name");
168     tableInput.getModel().addTableModelListener(new TableModelListener(){
169         @Override
170         public void tableChanged(TableModelEvent e) {
171             drawPanel.repaint();
172         }
173     });
174
175     UtilScrollPane scrollPaneInput = new UtilScrollPane(tableInput);
176     scrollPaneInput.setPreferredSize(new Dimension(50,85));
177     constraints.fill = GridBagConstraints.BOTH;
178     constraints.weightx = 1;
179     constraints.weighty = 1;
180     addComponent(topLayout, topPanel, scrollPaneInput, 2, 0, 1, 1);
181
182     //table for output
183     tableOutput = new TTable("Output name");
184     tableOutput.getModel().addTableModelListener(new TableModelListener(){
185         @Override
186         public void tableChanged(TableModelEvent e) {
187             drawPanel.repaint();
188         }
189     });
190
191     constraints.fill = GridBagConstraints.BOTH;
192     constraints.weightx = 1;
193     constraints.weighty = 1;
194     UtilScrollPane scrollPaneOutput = new UtilScrollPane(tableOutput);
195     scrollPaneOutput.setPreferredSize(new Dimension(50,85));
196     addComponent(topLayout, topPanel, scrollPaneOutput, 2, 1, 1, 1);
197
198     //add component title
199     constraints.fill = GridBagConstraints.BOTH;
200     constraints.weightx = 1;
201     constraints.weighty = 1;
202     constraints.anchor = GridBagConstraints.SOUTH;
203     addComponent(topLayout, topPanel, componentTitle, 3, 0, 1, 2);
204
205     //
206     constraints.fill = GridBagConstraints.HORIZONTAL;
207     constraints.weightx = 1;
208     constraints.weighty = 0;
209     topPanel.setBorder(BorderFactory.createLineBorder(Color.gray, 1));
210     addComponent(mainLayout, this, topPanel, 0, 0, 1, 1);
211
212     constraints.weightx = 3;
213     constraints.weighty = 3;
214     constraints.fill = GridBagConstraints.BOTH;
215
216     //draw graphics panel
217     drawPanel = new DrawPanel(tableInput.getModel(), tableOutput.getModel(), componentTitle);
218     drawPanel.setBackground(Color.WHITE);
219     drawPanel.setBorder(BorderFactory.createLineBorder(Color.gray, 1));

```

```

220     addComponent(mainLayout, this, drawPanel, 1, 0, 1, 1);
221
222     //preview button
223     JPanel okPanel = new JPanel(new FlowLayout());
224     okButton = new JButton("Add Component");
225     okButton.addActionListener(new ActionListener() {
226
227         @Override
228         public void actionPerformed(java.awt.event.ActionEvent e) {
229             try {
230                 dataPart = new DataPart(getTablePorts(tableInput),
231                                         getTablePorts(tableOutput),
232                                         new Point(300,300),
233                                         componentTitle.getText(),
234                                         (Graphics2D) DataPartBuilder.this.window.getGraphics());
235                 DataPartBuilder.this.window.addDataPart(dataPart);
236             } catch (IllegalArgumentException ex) {
237                 JOptionPane.showMessageDialog(DataPartBuilder.this, "Select valid type", "
238                 Invalid port type", JOptionPane.ERROR_MESSAGE);
239             }
240         });
241
242         okPanel.add(okButton);
243         okPanel.setBorder(BorderFactory.createLineBorder(Color.gray, 1));
244         constraints.fill = GridBagConstraints.HORIZONTAL;
245         constraints.weightx = 1;
246         constraints.weighty = 0;
247         addComponent(mainLayout, this, okPanel, 2, 0, 1, 1);
248     }
249     private void addComponent(GridBagLayout layout, Container p, Component comp, int r, int c, int
250     h, int w){
251         constraints.gridx = c;
252         constraints.gridy = r;
253         constraints.gridwidth = w;
254         constraints.gridheight = h;
255         layout.setConstraints(comp, constraints);
256         p.add(comp);
257         constraints.weightx = 0;
258         constraints.weighty = 0;
259         constraints.gridwidth = 1;
260         constraints.gridheight = 1;
261     }
262 }
263
264 @SuppressWarnings("serial")
265 class TModel extends DefaultTableModel {
266     ArrayList<TypeSelector> selectors = new ArrayList<>();
267     ArrayList<JButton> buttons = new ArrayList<>();
268     TModel(Object[] colNames) {
269         super(new Object[][] {}, colNames);
270     }
271     @Override
272     public boolean isCellEditable(int r, int c) {

```

```
273         return c == 0;
274     }
275     @Override
276     public void addRow(Object[] row) {
277         if (getRowCount() <= buttons.size()) {
278             int index = buttons.size();
279             JButton b = new JButton(row[1].toString());
280             b.addActionListener(new ActionListener() {
281                 @Override
282                 public void actionPerformed(ActionEvent e) {
283                     JDialog dialog = new JDialog();
284                     dialog.getContentPane().add(new UtilScrollPane(selectors.get(index)));
285                     dialog.setSize(500, 300);
286                     dialog.setVisible(true);
287                     dialog.addWindowListener(new WindowAdapter() {
288                         @Override
289                         public void windowClosing(WindowEvent e) {
290                             TModel.this.fireTableDataChanged();
291                         }
292                     });
293                 }
294             });
295             buttons.add(b);
296             selectors.add(new TypeSelector("Type"));
297         }
298         super.addRow(row);
299     }
300     public Type getTypeAt(int row) {
301         return selectors.get(row).getType();
302     }
303     @Override
304     public Object getValueAt(int r, int c) {
305         if (c == 1) {
306             return buttons.get(r);
307         } else {
308             return super.getValueAt(r, c);
309         }
310     }
311 }
312
313 class ButtonCellRenderer implements TableCellRenderer {
314
315     @Override
316     public Component getTableCellRendererComponent(JTable table, Object value,
317             boolean isSelected, boolean hasFocus, int row, int column) {
318         return (JButton)table.getValueAt(row, column);
319     }
320
321 }
322 @SuppressWarnings("serial")
323 class TTable extends JTable implements MouseListener {
324     public TTable(String column0) {
325         super(new TModel(new String[]{column0, "Type"}));
326         getColumnModel().getColumn(1).setCellRenderer(new ButtonCellRenderer());
327         addMouseListener(this);

```

```

328     }
329
330     @Override
331     public void mousePressed(MouseEvent e) {
332         int column = getColumnModel().getColumnIndexAtX(e.getX());
333         int row = e.getY()/getRowHeight();
334
335         if (row < getRowCount() && row >= 0 && column < getColumnCount() && column >= 0) {
336             Object value = this.getValueAt(row, column);
337             if (value instanceof JButton) {
338                 ((JButton) value).doClick();
339             }
340         }
341     }
342
343     @Override
344     public void mouseClicked(MouseEvent e) {
345     }
346     @Override
347     public void mouseReleased(MouseEvent e) {
348     }
349     @Override
350     public void mouseEntered(MouseEvent e) {
351     }
352     @Override
353     public void mouseExited(MouseEvent e) {
354     }
355 }
356 */

```

LISTING E.118: gui/temp/DataPartBuilder.java

```

1 package gui.temp;
2
3 import javax.swing.JPanel;
4
5 @SuppressWarnings("serial")
6 public class DrawPanel extends JPanel{
7     /*private static final long serialVersionUID = 1L;
8
9     private TableModel inputTableModel;
10    private TableModel outputTableModel;
11    private JTextField componentTitle;
12
13    public DrawPanel(TableModel im, TableModel om, JTextField componentTitle) {
14        inputTableModel = im;
15        outputTableModel = om;
16        this.componentTitle = componentTitle;
17    }
18
19    public void paintComponent(Graphics g){
20        Graphics2D g2 = (Graphics2D)g;
21        super.paintComponent(g2);
22
23        try {

```

```

24         DataPart dp = new DataPart(DataPartBuilder.getTableModelPorts(inputTableModel),
25             DataPartBuilder.getTableModelPorts(outputTableModel), new Point(20,100),
26             componentTitle.getText(), g2);
27         dp.paint(g2);
28     } catch (IllegalArgumentException e) {
29         g2.drawString("Select Type", 20, 100);
30     }
31     */
32     /*
33     Graphics2D g2 = (Graphics2D)g;
34     super.paintComponent(g2);
35
36     int inputs = inputTableModel.getRowCount();
37     int outputs = outputTableModel.getRowCount();
38
39     boxWidth = Math.max((int) (25 * 1.25 * Math.max(inputs, outputs)), minBoxWidth);
40
41     // Initialize title and box width.
42     String text = componentTitle.getText();
43     if (text.length() == 0)
44         text = " ";
45
46     TextLayout title = new TextLayout(text, g2.getFont(), g2.getFontRenderContext());
47
48     final int room = 6;
49     text = text.substring(0, text.length()-1);
50     while (title.getBounds().getWidth() > maxBoxWidth - room/2) {
51         text = text.substring(0, text.length()-1);
52         title = new TextLayout(text+"..", g2.getFont(), g2.getFontRenderContext());
53     }
54     boxWidth = Math.max((int)boxWidth, (int)title.getBounds().getWidth()+room);
55
56     divideInput = boxWidth/(inputs+1);
57     inputArrow = new InputArrow((int)pointBox.getX()+divideInput-diameter/2, (int)pointBox.
58     getY()-diameter*3-lineHeight+diameter/2, diameter, lineHeight);
59
60     divideoutput = boxWidth/(outputs+1);
61     outputArrow = new OutputArrow((int)pointBox.getX()+divideoutput-diameter/2, (int)pointBox.
62     getY()+boxHeight-diameter/2, diameter, lineHeight);
63
64     int xPosIn;
65     int xPosOut;
66     //draw border box
67     g2.setColor(Color.BLACK);
68     drawBoundBox(g2);
69     //draw box
70     drawRoundRect(g2, pointBox.getX(), pointBox.getY(), boxWidth, boxHeight, Color.WHITE);
71     //draw component title
72     title.draw(g2, (float)pointBox.getX() + boxWidth/2 - (int) (title.getBounds().getWidth()/
73     2), (float) (pointBox.getY() + boxHeight/2.0 + title.getBounds().getHeight()/2.0));
74
75     //draw input arrows
    for(int i=0; i<inputs; i++){
        inputArrow.setText(inputTableModel.getValueAt(i, 0).toString());
        inputArrow.paintArrow(g2);

```

```

76         xPosIn = inputArrow.getX();
77         inputArrow.setX(xPosIn+divideInput);
78     }
79     //draw output arrows
80     for(int j=0; j<outputs; j++){
81         outputArrow.setText(outputTableModel.getValueAt(j, 0).toString());
82         outputArrow.paintArrow(g2);
83         xPosOut = outputArrow.getX();
84         outputArrow.setX(xPosOut+divideoutput);
85     }
86     */
87 /*}*/
88 }
```

LISTING E.119: gui/temp/DrawPanel.java

```

1 package gui.temp;
2
3 import java.awt.BorderLayout;
4 import java.awt.Color;
5 import java.awt.Component;
6 import java.awt.ComponentOrientation;
7 import java.awt.Dimension;
8 import java.awt.Font;
9 import java.awt.event.ActionEvent;
10 import java.awt.event.ActionListener;
11
12 import javax.swing.ComboBoxEditor;
13 import javax.swing.JButton;
14 import javax.swing.JComboBox;
15 import javax.swing.JFrame;
16 import javax.swing.JList;
17 import javax.swing.JTextField;
18 import javax.swing.ListCellRenderer;
19 import javax.swing.event.EventListenerList;
20 import javax.swing.plaf.basic.BasicComboBoxUI;
21
22 import util.UtilFont;
23
24 class ColorComboBoxEditor implements ComboBoxEditor {
25     final protected JTextField editor;
26
27     protected EventListenerList listenerList = new EventListenerList();
28
29     public ColorComboBoxEditor(Integer initialValue) {
30         editor = new JTextField("Index " + initialValue);
31         editor.setPreferredSize(new Dimension(10,33));
32     }
33
34     public void addActionListener(ActionListener l) {
35         listenerList.add(ActionListener.class, l);
36     }
37
38     public Component getEditorComponent() {
39         return editor;
```

```
40     }
41
42     public Object getItem() {
43         return editor.getBackground();
44     }
45
46     public void removeActionListener(ActionListener l) {
47         listenerList.remove(ActionListener.class, l);
48     }
49
50     public void selectAll() {
51     }
52
53     public void setItem(Object newValue) {
54         Integer intVal = (Integer)newValue;
55         editor.setText("Index " + intVal);
56     }
57
58     protected void fireActionEvent(Color color) {
59         Object listeners[] = listenerList.getListenerList();
60         for (int i = listeners.length - 2; i >= 0; i -= 2) {
61             if (listeners[i] == ActionListener.class) {
62                 ActionEvent actionEvent = new ActionEvent(editor,
63                     ActionEvent.ACTION_PERFORMED, color.toString());
64                 ((ActionListener) listeners[i + 1])
65                     .actionPerformed(actionEvent);
66             }
67         }
68     }
69 }
70
71 class ComboBoxRenderer<T> extends JTextField implements ListCellRenderer<T> {
72     private static final long serialVersionUID = 1L;
73     private Font uhOhFont;
74
75     public ComboBoxRenderer() {
76         setOpaque(true);
77         setHorizontalAlignment(CENTER);
78         setPreferredSize(new Dimension(10,33));
79     }
80
81     /*
82      * This method finds the image and text corresponding to the selected value
83      * and returns the label, set up to display the text and image.
84      */
85     @SuppressWarnings("rawtypes")
86     public Component getListCellRendererComponent(JList list, Object value,
87             int index, boolean isSelected, boolean cellHasFocus) {
88         // Get the selected index. (The index param isn't
89         // always valid, so just use the value.)
90         int selectedIndex = ((Integer) value).intValue();
91
92         if (isSelected) {
93             setBackground(list.getSelectionBackground());
94             setForeground(list.getSelectionForeground());
```

```
95         } else {
96             setBackground(list.getBackground());
97             setForeground(list.getForeground());
98         }
99
100        setUhOhText("Index " + selectedIndex, list.getFont());
101        return this;
102    }
103
104    // Set the font and text when no image was found.
105    protected void setUhOhText(String uhOhText, Font normalFont) {
106        if (uhOhFont == null) { // lazily create this font
107            uhOhFont = normalFont.deriveFont(Font.ITALIC);
108        }
109       setFont(uhOhFont);
110       setText(uhOhText);
111    }
112 }
113
114 public class ComboBoxTest {
115     public static void main(String args[]) {
116         Integer intValues[] = {0,1};
117
118         JFrame frame = new JFrame("Color JComboBox");
119         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
120
121         MyComboBox<Integer> comboBox = new MyComboBox<>(intValues);
122         comboBox.setEditable(true);
123         comboBox.setRenderer(new ComboBoxRenderer<Integer>());
124         Integer initial = (Integer)comboBox.getSelectedItem();
125
126         ComboBoxEditor editor = new ColorComboBoxEditor(initial);
127         comboBox.setEditor(editor);
128         frame.add(comboBox, BorderLayout.NORTH);
129
130         frame.setSize(300, 200);
131         frame.setVisible(true);
132     }
133 }
134
135 class MyComboBoxUI extends BasicComboBoxUI {
136     @SuppressWarnings("serial")
137     @Override
138     protected JButton createArrowButton() {
139         JButton ret = new JButton() {
140             @Override
141             public int getWidth() {
142                 return 25;
143             }
144         };
145         ret.setFont(new Font(UtilFont.fontName, Font.BOLD, 16));
146         ret.setText("\u21d5");
147         return ret;
148     }
149 }
```

```
150
151 class MyComboBox<T> extends JComboBox<T> {
152     private static final long serialVersionUID = 1L;
153
154     public MyComboBox(T[] vals) {
155         super(vals);
156         this.setUI(new MyComboBoxUI());
157         this.applyComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);
158     }
159 }
```

LISTING E.120: gui/temp/ComboBoxTest.java

```
1 package gui;
2
3 import gui.ASTLoader.ASTLoaderException;
4 import gui.TabbedPane.InvalidTitleException;
5 import gui.dataflow.DataPartDialog;
6 import lang.ast.ASTVisitorCheck;
7 import lang.ast.ASTVisitorGen;
8
9 import java.awt.BorderLayout;
10 import java.awt.Color;
11 import java.awt.Component;
12 import java.awt.Dimension;
13 import java.awt.EventQueue;
14 import java.awt.FlowLayout;
15 import java.awt.Font;
16 import java.awt.GridBagConstraints;
17 import java.awt.GridBagLayout;
18 import java.awt.Insets;
19 import java.awt.Point;
20 import java.awt.event.ActionEvent;
21 import java.awt.event.ActionListener;
22 import java.awt.event.FocusEvent;
23 import java.awt.event.FocusListener;
24 import java.awt.event.MouseAdapter;
25 import java.awt.event.MouseEvent;
26 import java.awt.event.MouseListener;
27 import java.awt.event.WindowAdapter;
28 import java.awt.event.WindowEvent;
29 import java.awt.event.WindowListener;
30 import java.io.File;
31 import java.io.FileNotFoundException;
32 import java.io.IOException;
33 import java.io.PrintStream;
34 import java.util.List;
35
36 import javax.swing.BorderFactory;
37 import javax.swing.Box;
38 import javax.swingBoxLayout;
39 import javax.swing.DefaultListModel;
40 import javax.swing.Icon;
41 import javax.swing.JButton;
42 import javax.swing.JComboBox;
```

```
43 import javax.swing.JDialog;
44 import javax.swing.JFileChooser;
45 import javax.swing.JFrame;
46 import javax.swing.JLabel;
47 import javax.swing.JList;
48 import javax.swing.JMenu;
49 import javax.swing.JMenuBar;
50 import javax.swing.JMenuItem;
51 import javax.swing.JOptionPane;
52 import javax.swing.JPanel;
53 import javax.swing.JSeparator;
54 import javax.swing.JSplitPane;
55 import javax.swing.JTabbedPane;
56 import javax.swing.JTextField;
57 import javax.swing.JTree;
58 import javax.swing.ListSelectionModel;
59 import javax.swing.SwingConstants;
60 import javax.swing.event.ChangeEvent;
61 import javax.swing.event.ChangeListener;
62 import javax.swing.filechooser.FileNameExtensionFilter;
63 import javax.swing.tree.DefaultMutableTreeNode;
64 import javax.swing.tree.DefaultTreeModel;
65 import javax.swing.tree.TreeCellRenderer;
66 import javax.swing.tree.TreePath;
67 import javax.swing.tree.TreeSelectionModel;
68
69 import util.IconUtil;
70 import utilInputDialog;
71 import util.UtilButton;
72 import util.UtilFont;
73 import util.UtilLabel;
74 import util.UtilRegex;
75 import util.UtilScrollPane;
76
77 @SuppressWarnings("serial")
78 public class MainWindow extends JFrame {
79     private JMenuBar menuBar;
80     private JSplitPane splitPane;
81     private Explorer explorer;
82     private JTabbedPane tabbedPane;
83     private JMenu menuFile = new JMenu("File");
84     private JMenu menuInsert = new JMenu("Insert");
85     private JMenu menuEdit = new JMenu("Edit");
86     private JButton compileButton = new JButton();
87     private JMenuItem saveProjectMenuItem = new JMenuItem("Save project");
88     private Icon normalCompileIcon = IconUtil.createImageIcon("icons/compile-arrow-green.png");
89     private Icon pressedCompileIcon = IconUtil.createImageIcon("icons/compile-arrow-orange.png");
90     private JPanel leftPanel;
91     private JPanel leftNorthPanel;
92     private JFileChooser fileChooser = new JFileChooser() {
93         @Override
94         public void approveSelection() {
95             File f = getSelectedFile();
96             if(f.exists() && getDialogType() == SAVE_DIALOG) {
97                 int confirm = JOptionPane.showConfirmDialog(this,"Overwrite existing file?",
```

```
98             "Overwrite confirmation", JOptionPane.YES_NO_OPTION);
99         if (confirm != 0)
100             return;
101     }
102     super.approveSelection();
103 }
104 };
105 private DefaultListModel<ASTVisitorCheck.Problem> messagesTableModel;
106
107 public static final String PLATFORM_P10 = "Medarbejderportal",
108     PLATFORM_P20 = "Netbank";
109 private final static String[] platforms = {PLATFORM_P10, PLATFORM_P20};
110 private final static String[] groups = {"Normal", "Administrator"};
111
112 public static String[] getPlatforms() {
113     return platforms.clone();
114 }
115
116 public static String[] getUserGroups() {
117     return groups.clone();
118 }
119
120 private ControlFlowPanel getRootControlFlowPanel() {
121     DefaultTreeModel model = explorer.getTreeModel();
122     Object root = explorer.getTreeModel().getRoot();
123     if (model.getChildCount(root) == 0)
124         return null;
125     DefaultMutableTreeNode node = (DefaultMutableTreeNode)model.getChildAt(root, 0);
126     TabbedPane.ScrollPane scroll = (TabbedPane.ScrollPane)node.getUserObject();
127     ControlFlowPanel cfp = (ControlFlowPanel) scroll.getTabbedPane();
128     return cfp;
129 }
130
131 private class CompileListener implements ActionListener {
132     @Override
133     public void actionPerformed(ActionEvent e) {
134         if (compileButton.getIcon() == pressedCompileIcon)
135             return;
136
137         ControlFlowPanel cfp = getRootControlFlowPanel();
138         if (cfp == null)
139             return;
140
141         compileButton.setIcon(pressedCompileIcon);
142         compileButton.revalidate();
143         compileButton.repaint();
144
145         AST ast = new AST(cfp);
146         clearMessages();
147         ASTVisitorCheck validator = new ASTVisitorCheck();
148         List<ASTVisitorCheck.Problem> problems = validator.getProblems(ast.getRoot());
149         addMessages(problems);
150         if (problems.size() > 0) {
151             compileButton.setIcon(normalCompileIcon);
152             compileButton.revalidate();
```

```
153         compileButton.repaint();
154         return;
155     }
156
157     new Thread() {
158         @Override
159         public void run() {
160             try {
161                 ASTVisitorGen.compile(ast.getRoot());
162             } catch (IOException ex) {
163                 EventQueue.invokeLater(() -> {
164                     JOptionPane.showMessageDialog(MainWindow.this,
165                         ex.getMessage(), "Compilation error", JOptionPane.
166                         ERROR_MESSAGE);
167                 });
168             } finally {
169                 EventQueue.invokeLater(() -> {
170                     compileButton.setIcon(normalCompileIcon);
171                     compileButton.revalidate();
172                     compileButton.repaint();
173                 });
174             }
175         }.start();
176     }
177 }
178
179 private class SaveListener implements ActionListener {
180     @Override
181     public void actionPerformed(ActionEvent e) {
182         ControlFlowPanel cfp = getRootControlFlowPanel();
183         if (cfp == null)
184             return;
185
186         PrintStream stream = getFile(cfp);
187         if (stream != null) {
188             AST ast = new AST(cfp);
189             ast.print(stream);
190         }
191     }
192
193     private PrintStream getFile(ControlFlowPanel panel) {
194         File sel = fileChooser.getSelectedFile();
195         if (sel == null || sel.getName().equals("")) {
196             char[] title = (panel.getTitle()+"_bide").toCharArray();
197             title[0] = Character.toUpperCase(title[0]);
198             fileChooser.setSelectedFile(new File(new String(title)));
199         }
200         int returnVal = fileChooser.showSaveDialog(MainWindow.this);
201         if (returnVal == JFileChooser.APPROVE_OPTION) {
202             try {
203                 File file = fileChooser.getSelectedFile();
204                 if (!file.getName().endsWith(".bide"))
205                     file = new File(file.toString() + ".bide");
206                 return new PrintStream(file);
207             }
208         }
209     }
210 }
```

```
207         } catch (FileNotFoundException e) {
208             JOptionPane.showMessageDialog(MainWindow.this, "Directory does not exist.\nTry
209             again.",
210                     "Unable to save", JOptionPane.ERROR_MESSAGE);
211             return null;
212         }
213     } else {
214         return null;
215     }
216 }
217
218 private class OpenListener implements ActionListener {
219     @Override
220     public void actionPerformed(ActionEvent e) {
221         File file = getFile();
222         if (file == null) {
223             return;
224         }
225
226         tabbedPane.removeAll();
227         explorer.clear();
228         try {
229             ASTLoader.open(file, MainWindow.this);
230             saveMenuItem.setEnabled(true);
231             compileButton.setEnabled(true);
232             clearMessages();
233             ControlFlowPanel panel = getRootControlFlowPanel();
234             resetLeftNorthPanel(panel.getPlatform(), panel.getUserGroup());
235         } catch (ASTLoaderException e1) {
236             tabbedPane.removeAll();
237             explorer.clear();
238             JOptionPane.showMessageDialog(MainWindow.this, e1.getMessage(),
239                     "Unable to open project", JOptionPane.ERROR_MESSAGE);
240         }
241     }
242
243     private File getFile() {
244         int returnVal = fileChooser.showOpenDialog(MainWindow.this);
245         if (returnVal == JFileChooser.APPROVE_OPTION) {
246             File file = fileChooser.getSelectedFile();
247             String filePath = file.getAbsolutePath();
248             if (!filePath.endsWith(".bide")) {
249                 file = new File(filePath+".bide");
250             }
251             return file;
252         } else {
253             return null;
254         }
255     }
256 }
257
258 private class NewProjectDialog extends JDialog implements ActionListener, WindowListener {
259     private JComboBox<String> platformBox = new JComboBox<>(platforms);
260     private JComboBox<String> groupBox = new JComboBox<>(groups);
```

```
261     private JTextField titleField = new JTextField();
262
263     private String controlFlowTitle;
264     private String platform;
265     private String group;
266
267     public NewProjectDialog() {
268         platformBox.setSelectedItem(null);
269         platformBox.setFont(UtilFont.textFont);
270         groupBox.setSelectedItem(null);
271         groupBox.setFont(UtilFont.textFont);
272
273         JPanel okPanel = new JPanel(new BorderLayout());
274
275         JPanel contentsPanel = new JPanel(new GridBagLayout());
276         JPanel contents = new JPanel();
277         GridBagConstraints constraints = new GridBagConstraints();
278         GridBagLayout layout = new GridBagLayout();
279         contents.setLayout(layout);
280
281         constraints.anchor = GridBagConstraints.LINE_END;
282         constraints.fill = GridBagConstraints.NONE;
283         constraints.weightx = 0;
284         constraints.weighty = 0;
285         constraints.insets = new Insets(5, 5, 5, 5);
286         constraints.gridx = 0;
287         constraints.gridy = 0;
288         contents.add(new UtilLabel("Select platform"), constraints);
289
290         constraints.anchor = GridBagConstraints.CENTER;
291         constraints.fill = GridBagConstraints.HORIZONTAL;
292         constraints.weightx = 100;
293         constraints.gridx = 1;
294         contents.add(platformBox, constraints);
295
296         constraints.anchor = GridBagConstraints.LINE_END;
297         constraints.fill = GridBagConstraints.NONE;
298         constraints.weightx = 0;
299         constraints.gridx = 1;
300         constraints.gridy = 1;
301         contents.add(new UtilLabel("Select user group"), constraints);
302
303         constraints.anchor = GridBagConstraints.CENTER;
304         constraints.fill = GridBagConstraints.HORIZONTAL;
305         constraints.weightx = 100;
306         constraints.gridx = 1;
307         contents.add(groupBox, constraints);
308
309         titleField.setFont(UtilFont.textFont);
310         titleField.setPreferredSize(new Dimension(200, titleField.getPreferredSize().height));
311
312         constraints.anchor = GridBagConstraints.LINE_END;
313         constraints.fill = GridBagConstraints.NONE;
314         constraints.weightx = 0;
315         constraints.gridy = 2;
```

```
316     constraints.gridx = 0;
317     contents.add(new UtilLabel("Project title"), constraints);
318
319     constraints.anchor = GridBagConstraints.CENTER;
320     constraints.fill = GridBagConstraints.HORIZONTAL;
321     constraints.weightx = 100;
322     constraints.gridx = 1;
323     contents.add(titleField, constraints);
324
325     contentsPanel.add(contents);
326     okPanel.add(contentsPanel);
327     JPanel okBottom = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));
328     UtilButton ok = new UtilButton("Ok");
329     ok.addActionListener(this);
330     okBottom.add(ok);
331     UtilButton cancel = new UtilButton("Cancel");
332     cancel.addActionListener(new ActionListener() {
333         @Override
334         public void actionPerformed(ActionEvent e) {
335             NewProjectDialog.this.dispose();
336         }
337     });
338     okBottom.add(cancel);
339     okPanel.add(okBottom, BorderLayout.SOUTH);
340
341     getContentPane().add(okPanel);
342
343     pack();
344     setTitle("New project");
345     setLocationRelativeTo(MainWindow.this);
346     setModal(true);
347     addWindowListener(this);
348     setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
349     setVisible(true);
350 }
351
352 @Override
353 public void actionPerformed(ActionEvent arg0) {
354     String p = (String)platformBox.getSelectedItem();
355     if (p == null) {
356         JOptionPane.showMessageDialog(NewProjectDialog.this, "Please select platform",
357             "Unable to create new project", JOptionPane.ERROR_MESSAGE);
358         return;
359     }
360     String g = (String)groupBox.getSelectedItem();
361     if (g == null) {
362         JOptionPane.showMessageDialog(NewProjectDialog.this, "Please select user group",
363             "Unable to create new project", JOptionPane.ERROR_MESSAGE);
364         return;
365     }
366     String title = titleField.getText();
367     if (!UtilRegex.isIdentifier(title)) {
368         JOptionPane.showMessageDialog(NewProjectDialog.this, "Invalid title '"+title+"'",
369             "Unable to create new project", JOptionPane.ERROR_MESSAGE);
370         return;
```

```
371         }
372         platform = p;
373         group = g;
374         controlFlowTitle = title;
375         dispose();
376     }
377
378     @Override
379     public void windowClosing(WindowEvent e) {
380         int confirm = JOptionPane.showConfirmDialog(NewProjectDialog.this,
381             "Close without creating new project?", "Close confirmation",
382             JOptionPane.YES_NO_OPTION);
383         if (confirm == 0) {
384             dispose();
385         }
386     }
387
388     @Override
389     public void windowActivated(WindowEvent e) {}
390     @Override
391     public void windowClosed(WindowEvent e) {}
392     @Override
393     public void windowDeactivated(WindowEvent e) {}
394     @Override
395     public void windowDeiconified(WindowEvent e) {}
396     @Override
397     public void windowIconified(WindowEvent e) {}
398     @Override
399     public void windowOpened(WindowEvent e) {}
400 }
401
402     public static void validatePlatform(String p) throws InvalidTitleException {
403         for (String e : platforms) {
404             if (e.equals(p))
405                 return;
406         }
407         throw new InvalidTitleException("Unexpected platform '"+p+"'");
408     }
409
410     public static void validateUserGroup(String g) throws InvalidTitleException {
411         for (String e : groups) {
412             if (e.equals(g))
413                 return;
414         }
415         throw new InvalidTitleException("Unexpected user group '"+g+"'");
416     }
417
418     public MainWindow() {
419         //setExtendedState(getExtendedState() | JFrame.MAXIMIZED_BOTH);
420         setSize(800, 600);
421         setMenuBar();
422         setSplitPanes();
423         setFileChooser();
424         setCloseOperation();
425         setTitle("Bide");
```

```
426         setVisible(true);
427     }
428
429     private void clearMessages() {
430         for (int i = messagesTableModel.getSize() - 1; i >= 0; i--)
431             messagesTableModel.remove(i);
432     }
433
434     private void addMessages(List<ASTVisitorCheck.Problem> problems) {
435         for (ASTVisitorCheck.Problem p : problems) {
436             messagesTableModel.addElement(p);
437         }
438     }
439
440     private void setCloseOperation(){
441         setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
442         this.addWindowListener(new WindowAdapter() {
443             @Override
444             public void windowClosing(WindowEvent e) {
445                 int confirm = JOptionPane.showOptionDialog(MainWindow.this,
446                     "Do you want to exit?",
447                     "Exit confirmation", JOptionPane.YES_NO_OPTION,
448                     JOptionPane.QUESTION_MESSAGE, null, null, null);
449                 if (confirm == 0) {
450                     System.exit(0);
451                 }
452             }
453         });
454     }
455     private void setMenuBar(){
456         menuBar = new JMenuBar();
457
458         menuFile.setFont(UtilFont.textFont);
459         menuInsert.setFont(UtilFont.textFont);
460         menuEdit.setFont(UtilFont.textFont);
461
462         JMenuItem newProject = new JMenuItem("New project");
463         newProject.setFont(UtilFont.textFont);
464         menuFile.add(newProject);
465         newProject.addActionListener(new ActionListener() {
466             @Override
467             public void actionPerformed(ActionEvent e) {
468                 NewProjectDialog dialog = new NewProjectDialog();
469                 if (dialog.controlFlowTitle == null)
470                     return;
471                 String title = dialog.controlFlowTitle;
472                 String platform = dialog.platform;
473                 String group = dialog.group;
474                 ControlFlowPanel panel;
475                 try {
476                     panel = new ControlFlowPanel(MainWindow.this, title, platform, group, null);
477                 } catch (InvalidTitleException ex) {
478                     JOptionPane.showMessageDialog(MainWindow.this, ex.getMessage(),
479                         "Unexpected title", JOptionPane.ERROR_MESSAGE);
480                 }
481             }
482         });
483     }
484 }
```

```
481         }
482         tabbedPane.removeAll();
483         explorer.clear();
484         addExplorerItem(null, panel.getScrollPane());
485         saveProjectMenuItem.setEnabled(true);
486         compileButton.setEnabled(true);
487         clearMessages();
488         resetLeftNorthPanel(panel.getPlatform(), panel.getUserGroup());
489         fileChooser.setSelectedFile(new File(""));
490     }
491 });
492 saveProjectMenuItem.setFont(UtilFont.textFont);
493 saveProjectMenuItem.setEnabled(false);
494 menuFile.add(saveProjectMenuItem);
495 saveProjectMenuItem.addActionListener(new SaveListener());
496
497 JMenuItem openProject = new JMenuItem("Open project");
498 openProject.setFont(UtilFont.textFont);
499 menuFile.add(openProject);
500 openProject.addActionListener(new OpenListener());
501
502 compileButton.setBorderPainted(false);
503 compileButton.setMargin(new Insets(0, 5, 0, 5));
504 compileButton.setContentAreaFilled(false);
505 compileButton.setFocusPainted(false);
506 compileButton.setOpaque(false);
507 compileButton.setIcon(normalCompileIcon);
508 compileButton.addActionListener(new CompileListener());
509
510 menuInsert.setEnabled(false);
511 menuEdit.setEnabled(false);
512 compileButton.setEnabled(false);
513
514 menuBar.add(menuFile);
515 menuBar.add(menuInsert);
516 menuBar.add(menuEdit);
517 menuBar.add(compileButton);
518 this.setJMenuBar(menuBar);
519 }
520
521 private void addZoomMenuEditItems(TabbedPane panel) {
522     JMenuItem zoomIn = new JMenuItem("Zoom in");
523     zoomIn.setFont(UtilFont.textFont);
524     menuEdit.add(zoomIn);
525     zoomIn.addActionListener(new ActionListener() {
526         @Override
527         public void actionPerformed(ActionEvent e) {
528             panel.zoomIn();
529         }
530     });
531
532     JMenuItem zoomOut = new JMenuItem("Zoom out");
533     zoomOut.setFont(UtilFont.textFont);
534     menuEdit.add(zoomOut);
535     zoomOut.addActionListener(new ActionListener() {
```

```
536     @Override
537     public void actionPerformed(ActionEvent e) {
538         panel.zoomOut();
539     }
540 );
541 }
542
543 private void updateMenuBar() {
544     TabbedPane.ScrollPane panel = (TabbedPane.ScrollPane) tabbedPane.getSelectedComponent();
545     if (panel != null && panel.getTabbedPane() instanceof ControlFlowPanel) {
546         menuInsert.removeAll();
547         menuEdit.removeAll();
548
549         JMenuItem dataItem = new JMenuItem("New dataflow");
550         dataItem.setFont(UtilFont.textFont);
551         menuInsert.add(dataItem);
552         dataItem.addActionListener(new ActionListener() {
553             @Override
554             public void actionPerformed(ActionEvent e) {
555                 ControlFlowPanel cfp = (ControlFlowPanel)panel.getTabbedPane();
556                 String title = InputDialog.show(MainWindow.this, "Enter title of new dataflow"
557 );
558                 if (title != null)
559                     try {
560                         cfp.addNewDataflowControlPart(title);
561                     } catch (InvalidTitleException ex) {
562                         JOptionPane.showMessageDialog(MainWindow.this,
563                             ex.getMessage(), "Unexpected title", JOptionPane.ERROR_MESSAGE
564 );
565                     }
566                 });
567                 JMenuItem ifInteractionItem = new JMenuItem("If");
568                 ifInteractionItem.setFont(UtilFont.textFont);
569                 menuInsert.add(ifInteractionItem);
570                 ifInteractionItem.addActionListener(new ActionListener() {
571                     @Override
572                     public void actionPerformed(ActionEvent e) {
573                         ControlFlowPanel cfp = (ControlFlowPanel)panel.getTabbedPane();
574                         cfp.addNewIfRuntimeInteractionControlPart();
575                     }
576                 });
577                 JMenuItem stopItem = new JMenuItem("Stop");
578                 stopItem.setFont(UtilFont.textFont);
579                 menuInsert.add(stopItem);
580                 stopItem.addActionListener(new ActionListener() {
581                     @Override
582                     public void actionPerformed(ActionEvent e) {
583                         ControlFlowPanel cfp = (ControlFlowPanel)panel.getTabbedPane();
584                         cfp.addNewStopControlPart();
585                     }
586                 });
587                 JMenuItem failItem = new JMenuItem("Fail");
588                 failItem.setFont(UtilFont.textFont);
589                 menuInsert.add(failItem);
```

```
589         failItem.addActionListener(new ActionListener() {
590             @Override
591             public void actionPerformed(ActionEvent e) {
592                 ControlFlowPanel cfp = (ControlFlowPanel)panel.getTabbedPanel();
593                 cfp.addNewFailControlPart();
594             }
595         });
596         JMenuItem interactItem = new JMenuItem("Service interaction");
597         interactItem.setFont(UtilFont.textFont);
598         menuInsert.add(interactItem);
599         interactItem.addActionListener(new ActionListener() {
600             @Override
601             public void actionPerformed(ActionEvent e) {
602                 ControlFlowPanel cfp = (ControlFlowPanel)panel.getTabbedPanel();
603                 new InteractionPanel(MainWindow.this, cfp);
604             }
605         });
606
607         JMenuItem setTitleItem = new JMenuItem("Update controlflow title");
608         setTitleItem.setFont(UtilFont.textFont);
609         menuEdit.add(setTitleItem);
610         setTitleItem.addActionListener(new ActionListener() {
611             @Override
612             public void actionPerformed(ActionEvent e) {
613                 String title = InputDialog.show(MainWindow.this, "Enter new controlflow title"
614             );
615                 if (title != null)
616                     updateTitle(title, panel);
617             }
618         });
619         JMenuItem setPlatformItem = new JMenuItem("Change platform");
620         setPlatformItem.setFont(UtilFont.textFont);
621         menuEdit.add(setPlatformItem);
622         setPlatformItem.addActionListener(new ActionListener() {
623             @Override
624             public void actionPerformed(ActionEvent e) {
625                 String plat = (String) JOptionPane.showInputDialog(MainWindow.this,
626                             "Select platform", "Change platform", JOptionPane.PLAIN_MESSAGE,
627                             null, platforms, null);
628                 try {
629                     if (plat != null) {
630                         ControlFlowPanel p = getRootControlFlowPanel();
631                         p.setPlatform(plat);
632                         resetLeftNorthPanel(p.getPlatform(), p.getUserGroup());
633                     }
634                 } catch (InvalidTitleException e1) {
635                     JOptionPane.showMessageDialog(MainWindow.this, e1.getMessage(),
636                             "Unable to change platform", JOptionPane.ERROR_MESSAGE);
637                 }
638             }
639         });
640         JMenuItem setGroupItem = new JMenuItem("Change user group");
641         setGroupItem.setFont(UtilFont.textFont);
642         menuEdit.add(setGroupItem);
643         setGroupItem.addActionListener(new ActionListener() {
```

```

643         @Override
644         public void actionPerformed(ActionEvent e) {
645             String g = (String) JOptionPane.showInputDialog(MainWindow.this,
646                         "Select user group", "Change user group", JOptionPane.PLAIN_MESSAGE,
647                         null, groups, null);
648             try {
649                 if (g != null) {
650                     ControlFlowPanel p = getRootControlFlowPanel();
651                     p.setPlatform(g);
652                     resetLeftNorthPanel(p.getPlatform(), p.getUserGroup());
653                 }
654             } catch (InvalidTitleException e1) {
655                 JOptionPane.showMessageDialog(MainWindow.this, e1.getMessage(),
656                         "Unable to change user group", JOptionPane.ERROR_MESSAGE);
657             }
658         });
659     );
660     addZoomMenuEditItems(panel.getTabbedPane());
661
662     menuInsert.setEnabled(true);
663     menuEdit.setEnabled(true);
664 } else if (panel != null && panel.getTabbedPane() instanceof DataFlowPanel) {
665     menuInsert.removeAll();
666     menuEdit.removeAll();
667
668     JMenuItem openBuilder = new JMenuItem("Insert component");
669     openBuilder.setFont(UtilFont.textFont);
670     menuInsert.add(openBuilder);
671     openBuilder.addActionListener(new ActionListener() {
672         @Override
673         public void actionPerformed(ActionEvent e) {
674             new DataPartDialog(MainWindow.this, (DataFlowPanel) panel.getTabbedPane());
675         }
676     });
677
678     JMenuItem setTitleItem = new JMenuItem("Update dataflow title");
679     setTitleItem.setFont(UtilFont.textFont);
680     menuEdit.add(setTitleItem);
681     setTitleItem.addActionListener(new ActionListener() {
682         @Override
683         public void actionPerformed(ActionEvent e) {
684             String title = InputDialog.show(MainWindow.this, "Enter new dataflow title");
685             if (title != null)
686                 updateTitle(title, panel);
687         }
688     });
689     addZoomMenuEditItems(panel.getTabbedPane());
690
691     menuInsert.setEnabled(true);
692     menuEdit.setEnabled(true);
693 } else {
694     menuInsert.setEnabled(false);
695     menuEdit.setEnabled(false);
696 }
697 }
```

```

698
699     private void setFileChooser() {
700         FileNameExtensionFilter filter = new FileNameExtensionFilter("Bide project files", "bide")
701         ;
702         fileChooser.setFileFilter(filter);
703     }
704
705     private void setSplitPanes(){
706         explorer = new Explorer(new DefaultTreeModel(new DefaultMutableTreeNode()));
707         JPanel left = new JPanel(new BorderLayout());
708         left.add(explorer);
709
710         tabbedPane = new JTabbedPane();
711         tabbedPane.setTabLayoutPolicy(JTabbedPane.SCROLL_TAB_LAYOUT);
712         tabbedPane.addChangeListener(new ChangeListener() {
713             @Override
714             public void stateChanged(ChangeEvent e) {
715                 updateMenuBar();
716                 explorer.treeDidChange();
717             }
718         });
719
720         messagesTableModel = new DefaultListModel<>();
721         JList<ASTVisitorCheck.Problem> messagesTable = new JList<ASTVisitorCheck.Problem>(
722             messagesTableModel);
723         messagesTable.addFocusListener(new FocusListener() {
724             @Override
725             public void focusGained(FocusEvent arg0) {}
726             @Override
727             public void focusLost(FocusEvent arg0) {
728                 messagesTable.clearSelection();
729             }
730         });
731         messagesTable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
732         messagesTable.setLayoutOrientation(JList.VERTICAL);
733         messagesTable.addMouseListener(new MouseAdapter() {
734             @Override
735             public void mousePressed(MouseEvent e) {
736                 ASTVisitorCheck.Problem problem = messagesTable.getSelectedValue();
737                 if (problem == null)
738                     return;
739                 TabbedPane panel = getTabbedPaneForPath(problem.getPath());
740                 if (panel == null) {
741                     JOptionPane.showMessageDialog(MainWindow.this, "Cannot find dataflow",
742                         "Unable to open tab", JOptionPane.INFORMATION_MESSAGE);
743                     return;
744                 }
745                 openExplorerItem(panel);
746                 if (problem.getEndSequenceNumber() < 0)
747                     panel.scrollToComponent(problem.getStartSequenceNumber());
748                 else
749                     panel.scrollToPath(problem.getStartSequenceNumber(), problem.
getStartEventIndex(),
750                             problem.getEndSequenceNumber());
751             }
752         });

```

```

750     });
751     messagesTable.setFont(UtilFont.textFont);
752
753     JPanel messagesPanel = new JPanel(new BorderLayout());
754     JPanel labelPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 0, 0));
755     labelPanel.add(new UtilLabel("Problems"));
756     JPanel seperatorPanel = new JPanel(new BorderLayout());
757     seperatorPanel.add(new JSeparator(SwingConstants.HORIZONTAL), BorderLayout.SOUTH);
758     seperatorPanel.add(labelPanel);
759     messagesPanel.add(seperatorPanel, BorderLayout.NORTH);
760     messagesPanel.add(messagesTable);
761     UtilScrollPane messagesScroll = new UtilScrollPane(messagesPanel);
762
763     JSplitPane rightSplit = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
764         tabbedPane, messagesScroll);
765     rightSplit.setDividerLocation(0.90);
766     rightSplit.setResizeWeight(0.90);
767
768     UtilScrollPane scrollLeft = new UtilScrollPane(left);
769     leftPanel = new JPanel(new BorderLayout());
770     leftPanel.add(scrollLeft);
771     leftNorthPanel = new JPanel();
772     leftNorthPanel.setLayout(new BoxLayout(leftNorthPanel, BoxLayout.Y_AXIS));
773     leftNorthPanel.add(new UtilLabel(" "));
774     leftNorthPanel.add(new UtilLabel(" "));
775     leftNorthPanel.setBorder(BorderFactory.createEtchedBorder());
776     leftPanel.add(leftNorthPanel, BorderLayout.SOUTH);
777     splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
778         leftPanel, rightSplit);
779     splitPane.setResizeWeight(0);
780     splitPane.setDividerLocation(200);
781     getContentPane().add(splitPane, BorderLayout.CENTER);
782 }
783
784 private void resetLeftNorthPanel(String platform, String group) {
785     leftPanel.remove(leftNorthPanel);
786     leftNorthPanel = new JPanel();
787     GridBagConstraints constraints = new GridBagConstraints();
788     GridBagLayout layout = new GridBagLayout();
789     leftNorthPanel = new JPanel(layout);
790     leftNorthPanel.setBorder(BorderFactory.createEtchedBorder());
791
792     constraints.anchor = GridBagConstraints.LINE_START;
793     constraints.fill = GridBagConstraints.NONE;
794     constraints.weightx = 0;
795     constraints.weighty = 0;
796     constraints.gridx = 0;
797     constraints.gridy = 0;
798     leftNorthPanel.add(new UtilLabel("Platform: "), constraints);
799
800     constraints.weightx = 100;
801     constraints.gridx = 1;
802     constraints.fill = GridBagConstraints.HORIZONTAL;
803     leftNorthPanel.add(new UtilLabel(platform), constraints);
804

```

```
805     constraints.weightx = 0;
806     constraints.gridx = 1;
807     constraints.gridy = 0;
808     constraints.fill = GridBagConstraints.NONE;
809     leftNorthPanel.add(new UtilLabel("User group: "), constraints);
810
811     constraints.weightx = 100;
812     constraints.gridx = 1;
813     constraints.fill = GridBagConstraints.HORIZONTAL;
814     leftNorthPanel.add(new UtilLabel(group), constraints);
815
816     leftPanel.add(leftNorthPanel, BorderLayout.SOUTH);
817     leftPanel.revalidate();
818     leftPanel.repaint();
819 }
820
821 private TabbedPane getTabbedPaneForPath(List<String> path) {
822     return getTabbedPaneForPath(path, -1, (DefaultMutableTreeNode)explorer.getTreeModel().
823     getRoot());
824 }
825
826 private TabbedPane getTabbedPaneForPath(List<String> path, int index, DefaultMutableTreeNode
827     parent) {
828     DefaultTreeModel model = explorer.getTreeModel();
829     if (path.size()-1 == index) {
830         TabbedPane.ScrollPane scroll = (TabbedPane.ScrollPane)parent.getUserObject();
831         TabbedPane panel = scroll.getTabbedPane();
832         if (panel.getTitle().equals(path.get(index))) {
833             return panel;
834         }
835         return null;
836     }
837     int size = model.getChildCount(parent);
838     for (int i = 0; i < size; i++) {
839         DefaultMutableTreeNode node = (DefaultMutableTreeNode)model getChild(parent, i);
840         TabbedPane panel = getTabbedPaneForPath(path, index+1, node);
841         if (panel != null)
842             return panel;
843     }
844     return null;
845 }
846
847 public void removeExplorerItem(TabbedPane item) {
848     tabbedPane.remove(item.getScrollPane());
849     DefaultMutableTreeNode n = explorer.getNode(item.getScrollPane());
850     explorer.getTreeModel().removeNodeFromParent(n);
851 }
852
853 public void addExplorerItem(TabbedPane parent, TabbedPane item) {
854     addExplorerItem(parent.getScrollPane(), item.getScrollPane());
855 }
856
857 public void openExplorerItem(TabbedPane item) {
```

```
858         openExplorerItem(item.getScrollPane());
859     }
860
861     private void openExplorerItem(TabbedPane.ScrollPane panel) {
862         int index = tabbedPane.indexOfComponent(panel);
863         int selected = tabbedPane.getSelectedIndex();
864         if (index == -1 || index != selected) {
865             if (index == -1) {
866                 index = selected+1;
867                 addTab(panel);
868             }
869             tabbedPane.setSelectedIndex(index);
870             //updateMenuBar();
871         }
872     }
873
874     void addExplorerItem(TabbedPane.ScrollPane parent, TabbedPane.ScrollPane item) {
875         DefaultMutableTreeNode parentNode = explorer.getNode(parent);
876         int childCount = parentNode.getChildCount();
877         DefaultMutableTreeNode node = new DefaultMutableTreeNode(item);
878         explorer.getTreeModel().insertNodeInto(node, parentNode, parentNode.getChildCount());
879         if (parentNode == explorer.getTreeModel().getRoot()) {
880             explorer.getTreeModel().nodeStructureChanged(parentNode);
881         }
882         if (childCount == 0) {
883             TreePath path = new TreePath(explorer.getTreeModel().getPathToRoot(parentNode));
884             explorer.expandRow(explorer.getRowForPath(path));
885         }
886     }
887
888     private void updateTitle(String title, TabbedPane.ScrollPane panel) {
889         try {
890             panel.getTabbedPane().setTitle(title);
891             updateTab(panel);
892             DefaultMutableTreeNode node = explorer.getNode(panel);
893             explorer.getTreeModel().nodeChanged(node);
894         } catch (InvalidTitleException e) {
895             JOptionPane.showMessageDialog(this, e.getMessage(), "Unexpected title", JOptionPane.ERROR_MESSAGE);
896         }
897     }
898
899     private void updateTab(TabbedPane.ScrollPane comp) {
900         int index = tabbedPane.indexOfComponent(comp);
901
902         JPanel pnlTab = new JPanel();
903         BoxLayout tLay = new BoxLayout(pnlTab, BoxLayout.X_AXIS);
904         pnlTab.setLayout(tLay);
905         pnlTab.setOpaque(false);
906         JLabel lblIcon = new JLabel(comp.getTabbedPane().getIcon());
907         JLabel lblTitle = new JLabel(comp.getTabbedPane().getTitle());
908         lblTitle.setFont(UtilFont.textFont);
909         Dimension prefLbl = lblTitle.getPreferredSize();
910         if (prefLbl.width > 100) {
911             prefLbl.width = 100;
```

```
912         lblTitle.setPreferredSize(prefLbl);
913     }
914
915     JButton btnClose = new JButton("\u2716");
916     btnClose.setMargin(new Insets(0,0,0,0));
917     btnClose.setFont(new Font(UtilFont.fontName, Font.BOLD, 8));
918     btnClose.setPreferredSize(new Dimension(13,13));
919
920     pnlTab.add(lblIcon);
921     pnlTab.add(lblTitle);
922     JPanel btnPanel = new JPanel(new GridBagLayout());
923     btnPanel.setOpaque(false);
924     btnPanel.add(btnClose);
925     pnlTab.add(Box.createRigidArea(new Dimension(10,0)));
926     pnlTab.add(btnPanel);
927
928     tabbedPane.setTabComponentAt(index, pnlTab);
929
930     btnClose.addActionListener(new ActionListener() {
931         @Override
932         public void actionPerformed(ActionEvent e) {
933             tabbedPane.remove(comp);
934             //updateMenuBar();
935         }
936     });
937 }
938
939 private void addTab(TabbedPane.ScrollPane comp) {
940     int index = tabbedPane.getSelectedIndex();
941     if (index == -1) {
942         tabbedPane.addTab("", comp);
943     } else {
944         tabbedPane.insertTab("", null, comp, null, index+1);
945     }
946     updateTab(comp);
947 }
948
949 private class Explorer extends JTree implements MouseListener {
950     private DefaultTreeModel treeModel;
951
952     public Explorer(DefaultTreeModel treeModel) {
953         super(treeModel);
954         this.treeModel = treeModel;
955
956         setBackground(Color.white);
957         getSelectionModel().setSelectionMode(TreeSelectionModel.SINGLE_TREE_SELECTION);
958         setShowsRootHandles(true);
959         setToggleClickCount(0);
960         setCellRenderer(new ExplorerRenderer());
961         addMouseListener(this);
962         setRootVisible(false);
963     }
964
965     public void clear() {
966         treeModel.setRoot(new DefaultMutableTreeNode());
```

```
967     }
968
969     public DefaultTreeModel getTreeModel() {
970         return treeModel;
971     }
972
973     public DefaultMutableTreeNode getNode(TabbedPane.ScrollPane userObject) {
974         if (userObject == null)
975             return (DefaultMutableTreeNode) treeModel.getRoot();
976         return getNode(userObject, (DefaultMutableTreeNode) treeModel.getRoot());
977     }
978
979     private DefaultMutableTreeNode getNode(TabbedPane.ScrollPane userObject,
980     DefaultMutableTreeNode node) {
980         if (node.getUserObject() == userObject) {
981             return node;
982         }
983         for (int i = 0; i < node.getChildCount(); i++) {
984             DefaultMutableTreeNode ret = (DefaultMutableTreeNode) node.getChildAt(i);
985             ret = getNode(userObject, ret);
986             if (ret != null) {
987                 return ret;
988             }
989         }
990         return null;
991     }
992
993     @Override
994     public void mousePressed(MouseEvent e) {
995         Point point = e.getPoint();
996         if (!e.isPopupTrigger() && getPathForLocation(point.x, point.y) != null) {
997             DefaultMutableTreeNode node =
998                 (DefaultMutableTreeNode) getLastSelectedPathComponent();
999             if (node == null)
1000                 return;
1001             TabbedPane.ScrollPane panel = (TabbedPane.ScrollPane) node.getUserObject();
1002             openExplorerItem(panel);
1003         }
1004     }
1005
1006     @Override
1007     public void mouseReleased(MouseEvent e) {}
1008     @Override
1009     public void mouseClicked(MouseEvent e) {}
1010     @Override
1011     public void mouseEntered(MouseEvent e) {}
1012     @Override
1013     public void mouseExited(MouseEvent e) {}
1014 }
1015
1016     private class ExplorerRenderer implements TreeCellRenderer {
1017         @Override
1018         public Component getTreeCellRendererComponent(JTree tree, Object value,
1019             boolean selected, boolean expanded, boolean leaf, int row,
1020             boolean hasFocus) {
```

```

1021     Object userData = ((DefaultMutableTreeNode)value).getUserObject();
1022     JPanel p = new JPanel();
1023     p.setBackground(Color.white);
1024     p.setLayout(new BoxLayout(p, BoxLayout.X_AXIS));
1025     if (userData instanceof TabbedPane.ScrollPane) {
1026         TabbedPane.ScrollPane tp = (TabbedPane.ScrollPane) userData;
1027         Icon icon = tp.getTabbedPane().getIcon();
1028         JLabel iconLabel = new JLabel(icon);
1029         JLabel titleLabel = new JLabel(tp.getTabbedPane().getTitle());
1030         titleLabel.setFont(UtilFont.textFont);
1031         p.add(iconLabel);
1032         p.add(titleLabel);
1033         if (tabbedPane.getSelectedComponent() == tp) {
1034             p.setBackground(new Color(0xD3, 0xD3, 0xD3));
1035             //titleLabel.setOpaque(true);
1036         }
1037     } else {
1038         p.add(new JLabel(value.toString()));
1039     }
1040     p.setPreferredSize(new Dimension(p.getPreferredSize().width, 18));
1041     return p;
1042 }
1043 }
1044 }
```

LISTING E.121: gui/MainWindow.java

```

1 package gui;
2
3 import java.awt.BorderLayout;
4 import java.awt.FlowLayout;
5 import java.awt.event.ActionEvent;
6 import java.awt.event.ActionListener;
7 import java.awt.event.WindowEvent;
8 import java.awt.event.WindowListener;
9 import java.util.ArrayList;
10 import java.util.List;
11 import java.util.function.Consumer;
12
13 import javax.swing.JComboBox;
14 import javax.swing.JDialog;
15 import javax.swing.JOptionPane;
16 import javax.swing.JPanel;
17
18 import lang.type.RecordColumn;
19 import lang.type.TypeList;
20 import lang.type.TypeString;
21 import util.UtilFont;
22 import util.UtilLabel;
23
24 @SuppressWarnings("serial")
25 public class InteractionPanel extends JDialog implements WindowListener, ActionListener {
26     private final static String[] p10Interactions = {
27         "Message",
28         "Table",
```

```
29         "Unary_input",
30         "Binary_input",
31         "Binary_button"
32     };
33     private final static String[][] p10Events = {
34         {"Success"},
35         {"Success"},
36         {"Success"},
37         {"Success"},
38         {"Button1", "Button2"}
39     };
40     private final static List<List<RecordColumn>> p10Inputs = new ArrayList<>();
41     private final static List<List<RecordColumn>> p10Outputs = new ArrayList<>();
42
43     private final static String[] p20Interactions = {
44         "Message",
45         "Table",
46         "Unary_input",
47         "Binary_input",
48         "Binary_button"
49     };
50     private final static String[][] p20Events = {
51         {"Success"},
52         {"Success"},
53         {"Success"},
54         {"Success"},
55         {"Button1", "Button2"}
56     };
57     private final static List<List<RecordColumn>> p20Inputs = new ArrayList<>();
58     private final static List<List<RecordColumn>> p20Outputs = new ArrayList<>();
59
60     static {
61         ArrayList<RecordColumn> input0 = new ArrayList<>();
62         input0.add(new RecordColumn("User", new TypeString()));
63         input0.add(new RecordColumn("Message", new TypeString()));
64         p10Inputs.add(input0);
65         p20Inputs.add(input0);
66         ArrayList<RecordColumn> input01 = new ArrayList<>();
67         input01.add(new RecordColumn("User", new TypeString()));
68         input01.add(new RecordColumn("Header", new TypeString()));
69         input01.add(new RecordColumn("Table", new TypeList(new TypeString())));
70         p10Inputs.add(input01);
71         p20Inputs.add(input01);
72         ArrayList<RecordColumn> input02 = new ArrayList<>();
73         input02.add(new RecordColumn("User", new TypeString()));
74         input02.add(new RecordColumn("Message", new TypeString()));
75         input02.add(new RecordColumn("Label", new TypeString()));
76         p10Inputs.add(input02);
77         p20Inputs.add(input02);
78         ArrayList<RecordColumn> input1 = new ArrayList<>();
79         input1.add(new RecordColumn("User", new TypeString()));
80         input1.add(new RecordColumn("Message", new TypeString()));
81         input1.add(new RecordColumn("Label1", new TypeString()));
82         input1.add(new RecordColumn("Label2", new TypeString()));
83         p10Inputs.add(input1);
```

```
84     p20Inputs.add(input1);
85     ArrayList<RecordColumn> input2 = new ArrayList<>();
86     input2.add(new RecordColumn("User", new TypeString()));
87     input2.add(new RecordColumn("Message", new TypeString()));
88     input2.add(new RecordColumn("Button_label1", new TypeString()));
89     input2.add(new RecordColumn("Button_label2", new TypeString()));
90     p10Inputs.add(input2);
91     p20Inputs.add(input2);
92
93     ArrayList<RecordColumn> outputs0 = new ArrayList<>();
94     p10Outputs.add(outputs0);
95     p20Outputs.add(outputs0);
96     ArrayList<RecordColumn> outputs01 = new ArrayList<>();
97     p10Outputs.add(outputs01);
98     p20Outputs.add(outputs01);
99     ArrayList<RecordColumn> outputs02 = new ArrayList<>();
100    outputs02.add(new RecordColumn("Input", new TypeString()));
101    p10Outputs.add(outputs02);
102    p20Outputs.add(outputs02);
103    ArrayList<RecordColumn> outputs1 = new ArrayList<>();
104    outputs1.add(new RecordColumn("Input1", new TypeString()));
105    outputs1.add(new RecordColumn("Input2", new TypeString()));
106    p10Outputs.add(outputs1);
107    p20Outputs.add(outputs1);
108    ArrayList<RecordColumn> outputs2 = new ArrayList<>();
109    p10Outputs.add(outputs2);
110    p20Outputs.add(outputs2);
111 }
112
113 private JComboBox<String> platforms;
114 private JPanel platformPanel;
115 private ControlFlowPanel controlFlowPanel;
116 private Consumer<String> closeAction;
117
118 public InteractionPanel(MainWindow mainWin, ControlFlowPanel panel) {
119     controlFlowPanel = panel;
120
121     platforms = new JComboBox<String>(MainWindow.getPlatforms());
122     platforms.setSelectedItem(null);
123     platforms.addActionListener(this);
124     platforms.setFont(UtilFont.textField);
125
126     JPanel platformsPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));
127     platformsPanel.add(new UtilLabel("Platform"));
128     platformsPanel.add(platforms);
129
130     getContentPane().add(platformsPanel, BorderLayout.NORTH);
131
132     setSize(300, 150);
133     setTitle("Insert service interaction");
134     setLocationRelativeTo(mainWin);
135     setModal(true);
136     addWindowListener(this);
137     setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
138     setVisible(true);
```

```
139     }
140
141     private void addP10Interaction(String inter) {
142         for (int i = 0; i < p10Interactions.length; i++) {
143             if (inter.equals(p10Interactions[i])) {
144                 controlFlowPanel.addNewInteractionControlPart(p10Interactions[i],
145                     (String)platforms.getSelectedItem(), p10Events[i],
146                     p10Inputs.get(i), p10Outputs.get(i));
147             }
148         }
149     }
150     throw new RuntimeException("unknown P10 interaction " + inter);
151 }
152
153     private void addP20Interaction(String inter) {
154         for (int i = 0; i < p20Interactions.length; i++) {
155             if (inter.equals(p20Interactions[i])) {
156                 controlFlowPanel.addNewInteractionControlPart(p20Interactions[i],
157                     (String)platforms.getSelectedItem(), p20Events[i],
158                     p20Inputs.get(i), p20Outputs.get(i));
159             }
160         }
161     }
162     throw new RuntimeException("unknown P10 interaction " + inter);
163 }
164
165     @Override
166     public void actionPerformed(ActionEvent arg0) {
167         closeAction = null;
168         String p = (String)platforms.getSelectedItem();
169         if (p.equals(MainWindow.PLATFORM_P10)) {
170             setPlatformP10Panel();
171         } else if (p.equals(MainWindow.PLATFORM_P20)) {
172             setPlatformP20Panel();
173         } else {
174             throw new RuntimeException("unexpected platform selected " + p);
175         }
176     }
177
178     private void setPlatformP10Panel() {
179         if (platformPanel != null) {
180             getContentPane().remove(platformPanel);
181         }
182         platformPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));
183         platformPanel.add(new UtilLabel("Interaction"));
184         JComboBox<String> box = new JComboBox<>(p10Interactions);
185         box.setFont(UtilFont.textField);
186         box.setSelectedItem(null);
187         box.addActionListener((e) -> {
188             closeAction = (s) -> addP10Interaction((String)box.getSelectedItem());
189         });
190         platformPanel.add(box);
191         getContentPane().add(platformPanel);
192         revalidate();
193         repaint();
```

```
194     }
195
196     private void setPlatformP20Panel() {
197         if (platformPanel != null) {
198             getContentPane().remove(platformPanel);
199         }
200         platformPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 5, 5));
201         platformPanel.add(new UtilLabel("Interaction"));
202         JComboBox<String> box = new JComboBox<>(p20Interactions);
203         box.setFont(UtilFont.textFont);
204         box.setSelectedItem(null);
205         box.addActionListener((e) -> {
206             closeAction = (s) -> addP20Interaction((String)box.getSelectedItem());
207         });
208         platformPanel.add(box);
209         getContentPane().add(platformPanel);
210         revalidate();
211         repaint();
212     }
213
214     @Override
215     public void windowClosing(WindowEvent arg0) {
216         if (closeAction == null) {
217             int confirm = JOptionPane.showConfirmDialog(this, "Close without adding interaction?", "Close confirmation", JOptionPane.YES_NO_OPTION);
218             if (confirm == 0)
219                 dispose();
220             else {
221                 closeAction.accept("");
222                 dispose();
223             }
224         }
225     }
226
227     @Override
228     public void windowActivated(WindowEvent arg0) {}
229     @Override
230     public void windowClosed(WindowEvent arg0) {}
231     @Override
232     public void windowDeactivated(WindowEvent arg0) {}
233     @Override
234     public void windowDeiconified(WindowEvent arg0) {}
235     @Override
236     public void windowIconified(WindowEvent arg0) {}
237     @Override
238     public void windowOpened(WindowEvent arg0) {}
239 }
```

LISTING E.122: gui/InteractionPanel.java

Bibliography

- [1] Ian F. Sommerville. Software Engineering, 2010.
- [2] Business Process Model and Notation from Wikipedia, . URL
https://en.wikipedia.org/wiki/Business_Process_Model_and_Notation.
- [3] Activity Diagram from Wikipedia, . URL
https://en.wikipedia.org/wiki/Activity_diagram.
- [4] Business Process Execution Language from Wikipedia, . URL
https://en.wikipedia.org/wiki/Business_Process_Execution_Language.
- [5] Oracle BPEL Process Manager example, . URL https://docs.oracle.com/cd/E23943_01/integration.1111/e10231/adptr_db.htm.
- [6] Oracle BPEL Process Manager, . URL <http://www.oracle.com/technetwork/middleware/bpel/overview/index.html>.
- [7] LabVIEW Description, . URL <http://www.ni.com/labview/why/>.
- [8] LabVIEW from Wikipedia, . URL <https://en.wikipedia.org/wiki/LabVIEW>.
- [9] LabVIEW Video Example, . URL <http://www.ni.com/academic/students/learn-labview/execution-structures/>.
- [10] SQL Server Integration Services from MSDN, . URL
<https://msdn.microsoft.com/en-us/library/ms141026.aspx>.
- [11] SQL Server Integration Services Video Example, . URL
https://www.youtube.com/watch?v=VZrlXwM0_Pg.
- [12] Business Process Model and Notation, 2011. URL
<http://www.omg.org/spec/BPMN/2.0/PDF/>.
- [13] Oasis BPEL Standard. URL
<http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf>.

Bibliography

- [14] SQL Server Integration Services Expressions, . URL
<https://msdn.microsoft.com/da-dk/library/ms141671.aspx>.
- [15] Glasgow Haskell Compiler. URL <https://www.haskell.org/ghc/>.
- [16] Camunda Home Page, . URL <https://camunda.org>.
- [17] Camunda REST API, . URL
<https://docs.camunda.org/manual/7.5/reference/rest/overview/>.