The Visual Service Design Tool
Version 1.1.1
Manual

Tobias Küster
DAI-Labor, TU Berlin
tobias.kuester@dai-labor.de

2008 / 11

Contents

1 Introduction

1.1 Motivation for Developing the VSDT

The goal of process modelling, as of Model Driven Engineering in general, is to provide an abstract view on systems, and to design those systems in a language and platform independent way. For that purpose the Business Process Modelling Notation (BPMN) [6] has been standardised by the Object Management Group. It can be understood intuitively by all business partners, even those who have great knowledge in their domain but do not know too much about Service Oriented Architecture (SOA) or programming in general. At the same time, BPMN is formal enough to provide a basis for the later implementation and refinement of the business process. Given a respective mapping, a BPMN diagram can be used for generating readily executable code from it. A brief introduction to BPMN is given for instance in [8] and in Appendix A of this manual.

Today, the Business Process Modelling Notation and the specified mapping to the Business Process Execution Language (BPEL) are supported by a growing number of tools. However, the problem with the majority of existing tools is that while they do provide the usual transformations from BPMN to BPEL, they are focused only on this one aspect of BPMN. Often the editors and even the underlying metamodels are adapted to BPEL in many ways. While this may be desired in order to provide highest possible usability and to support the user in the creation of executable BPEL code, the consequence is that business process diagrams created with these tools can neither be transformed to other executable languages, nor can the process model be used with other tools that might provide different transformations. Thus, while process modelling and BPMN should be independent of a specific executable language, the *tools* are not.

The solution to this problem is to keep both the underlying BPMN metamodel and the diagram editor free from influences from the BPEL world and to use pure BPMN instead, so that diagrams created with such a tool will be truly independent of any concrete language — apart from what influenced the BPMN specification in the first place. Based on this, several mappings to different target languages can be implemented and integrated into the editor as plugins, which may also contribute to the editor in order to support the business architect with language-specific support.

Following this approach, the *Visual Service Design Tool* (VSDT) has been implemented as an Eclipse plugin, inherently providing the necessary modularity. For the export of BPMN diagrams to executable languages a transformation framework has been designed. The actual transformations have been subdivided in distinct stages, so that significant parts of it are reusable, e.g. the challenging transformation of the control flow. Thus the actual mapping to a given language can be integrated in a very straightforward way. While the usual mapping from BPMN to BPEL has been realised as a

proof of concept, the main intent behind the VSDT is to provide a transformation from business processes to multi-agent systems such as the JIAC language family [7].[1] The respective mappings are currently under development. Our ultimate goal is to provide transformations not only in different, but also in heterogeneous systems — just like they are used in the real business world.

1.2 The Visual Service Design Tool

The first version of the VSDT has been developed as a diploma thesis [4] in the course of the *Service Centric Home* (SerCHo) project at TU Berlin in early 2007. As the work continued it matured to a feature-rich BPMN editor with an extensible transformation framework and has already been used in a number of service orchestration scenarios in the context of a smart home environment.

1.2.1 The Metamodel

The BPMN specification [6] describes in detail how the several nodes and connections constituting a BPMN diagram have to look, in which context they may be used and what attributes they have to provide. However, it does neither give a formal definition of the syntax to be used for the metamodel, nor an interchange format, e. g. using an XML Schema Definition (XSD). Thus the editor's metamodel had to be derived from the informal descriptions in the specification. As it was our main concern to keep as close to the specification as possible, almost every attribute and each constraint given in the specification has been incorporated into the metamodel, allowing the creation of any legal business process diagram. Still, some attributes have not been adopted in the metamodel: For instance the possibility to model nested or even crossing Lanes has been dropped, as it turned out that this feature seems to be virtually never used in practical business process design. Further, redundant attributes, such as the Gateway's `defaultGate` attribute, are emulated using getter methods to prevent inconsistency in the diagram model.

1.2.2 The BPMN Editor

Like many others, the VSDT editor has been created using the Eclipse Graphical Modelling Framework (GMF), automatically equipping the editor with numerous features, such as support for the Eclipse properties, outline and problem view and unlimited undo and redo, just to name a few. Being embedded in the Eclipse workbench, the editor is easy to use while at the same time providing a powerful tool for professional business architects and service developers.

While GMF provides a solid basis for the editor, several customisations have been made to the code, further improving the editor's overall usability and supporting the creation of new business processes. For example, the generated property tables have

[1] http://www.jiac.de/

been supplemented with custom-made sheets, in which the several attributes are more clearly arranged. For managing the non-visual elements given in the BPMN specification, such as Properties, Messages and Assignments, a number of clear and uniform dialogs has been created. The various constraints given in the specification were translated to several audit constraints used to validate a given business process diagram.

As already mentioned, the VSDT was designed to be a pure BPMN editor and independent of BPEL, so the business process diagrams can be transformed to other languages, too, given the respective export plugins. Of course, the downside of this approach is that the editor lacks built-in support for BPEL, e.g. the editor itself does not validate an expression given in the diagram to conform to the BPEL syntax. However, it is possible to supplement the editor with additional plugins, which can contribute e.g. to the property sheets or provide whole new views with language-specific functionality.

One example of how the VSDT can be extended with features specific to a certain target language — in this case: BPEL — is the RSD View, which can bee seen in Figure 3.1, too: A client for the *Rich Service Directory*, a special kind of Web service repository. Using the RSD View, existing Web services that have been registered at the RSD server can be inspected and imported into the diagram. In the process, an Implementation object is created for the Web service as well as a set of Message objects, matching the service's input parameters and result. Optionally, also a new Pool will be created for the service, which can be connected to the currently selected Activity via a pair of Message Flows. Further, the Implementation and the Message objects will be associated to the Activity and its type will be set to SERVICE. Thus, the orchestration of existing Web services in a BPEL process can be simplified greatly. Similar features can be created for other target languages, too.

Once the business process diagram is completed it can be validated and exported to an executable language, such as BPEL. As the VSDT is intended to provide export features to arbitrary target languages, and to support the tool smiths in the creation of these features, we have created an elaborate export framework. For each of these features, being realized as additional Eclipse plugins, an individual wizard can be made available in the Export menu.

For more information regarding the Visual Service Design Tool, please refer to [5].

2 Setup

In the following the installation process for the VSDT will be explained. As the VSDT is a plugin to the Eclipse IDE, the installation process is subdivided in several steps:

1. Install Eclipse
2. Install Dependencies
3. Install VSDT

Note that for running the VSDT Version 5 of the Java environement is required.
Note that the versions of the dependencies may vary depending on the version of the VSDT. The following applies to version 1.1.1of the VSDT.

2.1 Installing Eclipse

For using the Visual Service Design Tool you need the Eclipse IDE which can be downloaded from `http://www.eclipse.org` for different operating systems.
The VSDT 1.1.1works with Eclipse 3.4 "Ganymede".

2.2 Installing Dependencies

The VSDT depends on a number of other plugins, maily the Graphical Modeling Framework GMF, which again will have some dependencies on its own. You can get GMF at `http://www.eclipse.org/gmf`. However, it is recommended to use the Eclipse Update feature, as in that case all the dependencies and the dependencies of the dependencies will automatically be installed, too.

For installing the plugins, make sure you are allowed to write in the Eclipse program folder, especially when running Linux you should start that instance of Eclipse with `sudo`. In Eclipse, select *Help* from the menu, then *Software Updates...*. In the *Available software* tab, browse the *Ganymede* update site and select the *Graphical Modeling Framework* from one of the lists. Now click on *Install* to automatically check all required features, like EMF and GEF, and to continue through the installation process. After that you might have to restart Eclipse. If the installation was successfull the new features should appear in the *Installed Software* tab of the same dialog.
The VSDT 1.1.1was created with GMF 2.1.

2.3 Installing the VSDT

Once Eclipse and GMF are set up you can install VSDT. The Visual Service Design Tool consists of the following features:

- *VSDT: Visual Service Design Tool* This is the core component of the VSDT: The visual BPMN editor.
- *VSDT: RSD Integration* This feature contributes an additional view to the BPMN editor, making the reuse of existing Web services easier.
- *VSDT: Transformation Base* This is an enabling feature for the several export features, providing the common functionality and look and feel for the transformations.
- *VSDT: BPMN - BPEL Transformation* This feature provides the export to executable BPEL code.[1]
- *VSDT: BPMN - ... Transformation* Further export and import features will be released over time, providing transformations to other languages, such as JIAC agents.

The latest version of the VSDT can be found at the VSDT Update Site:

`http://repositories.dai-labor.de/sites/vsdt/`

Download the VSDT as an archive or add the VSDT Update Site in the *Software Updates* dialog (see last section), using the *Add Site...* button, and use the update feature to install the plugins.

Further to that, the VSDT can be obtained from the following sites:

`http://energy.dai-labor.de/`

[1] In future releases, the Transformation plugins may provide Import functionality, as well.

3 The Easy Service Creation Perspective

This section will briefly introduce those parts of the Eclipse GUI that are relevant for the work with the Visual Service Design Tool. These views are aggregated in the Easy Service Creation perspective, which an be selected via the Menu *Window* → *Perspective*. Figure 3.1 is showing a screenshot of the Visual Service Design Tool featuring most of the relevant views. In the following each of the views will be introduced briefly.

3.1 BPMN Editor Views

The editor window is shown automatically when opening a file in the Navigator view. Depending on the PlugIns currently installed this can be a plain text editor, a browser, an elaborate code editor or some sort of graphical editor. For the Visual Service Design Tool there are two editors available: A visual editor showing the BPMN graph and a tree editor reflecting the internal structure.

The Graphical Editor This is the primary editor when working with the VSDT, which is opened when the diagram file is clicked. On the side there is a palette with the nodes and connections. For placing a node on the canvas or inside a compartment of another node (e.g. a Pool or a Sub Process) click the icon in the palette and then click again on the canvas. For drawing connections, click on the first node, draw the connection to the second node and release the mouse button. Note that nodes and connections can not be drawn arbitrarily, but have to follow the BPMN syntax, e.g. a Task can only be drawn inside a Pool and a Sequence Flow can only connect Flow Object within the same Pool.

The Tree Editor The tree editor can be useful for managing and editing those parts of the Business Process Diagram that do not have a graphical representation.[1] Note that the tree editor is more powerful than the graphical editor, and the diagram might be invalidated when doing certain operations in the tree editor. Especially, the tree editor should *not* be used for creating any elements that *do* have a graphical representation in the diagram, as this representation will not be created along with the element. In case the diagram file is broken, it can be recreated from the model file by right-clicking it and selecting the respective menu item.

The Text Editor Both the diagram and the model file can be opened with any text editor and edited as XML. While this can be helpful for adapting existing models to

[1] Although in general it is not necessary to use the tree editor, as the graphical editor provides means for editing non-graphical elements, too.

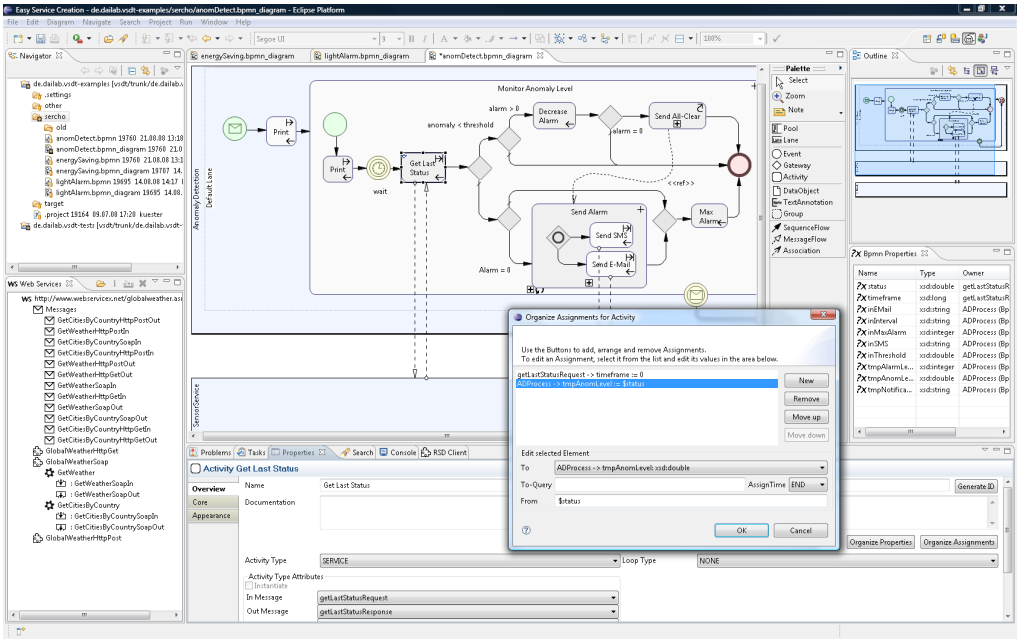

Figure 3.1: Clockwise: Graphical Editor (with enabled colors and markers), Outline View, BPMN-Properties View, Organize-Assignments Dialog, Property View, Web Service View, Navigator.

changes in the metamodel of a newer version of the VSDT, you should in general avoid editing the files XML sources, as this can render them unreadable for the other editors.

3.2 General-purpose Eclipse Views

In the following those standard views of the Eclipse IDE will be introduced, that are relevant for the work with the VSDT.

The Navigator Here the user can manage his projects and create and delete files. Note that Eclipse provides different similar views for managing files, e.g. the Navigator, Project Explorer or the Package Explorer, each providing slightly different features.

The Properties View Although some attributes, like an element's name, can be edited in the graphical editor view as well, for most other attributes the properties view will be needed, where all the attributes relevant to the user can be inspected and edited. Of course, each change done in the properties view can be undone and redone and the editor will be immediately updated. There are two tabs available in the properties view: The *Core* tab provides a table showing the attributes in categories and in alphabetic order. The *Overview* tab provides a clearer look, grouping the attributes and arranging them by relevance in two columns. Additionally, the Outline tab features a number of

buttons, providing access to additional dialogs for managing e.g. an Activity's Properties and Assignments.

The Outline This view provides a short outline of the current editor's content. In case of a graphical editor, like the VSDT, this can be a miniature view of the entire diagram, and in case of a tree editor an additional tree view for easier navigation.

The Problem View This view lists all the problems that have been found in the model, subdivided in errors and warnings. By double-clicking one of the items the editor will focus on the element the problem occurred on (for refreshing the errors shown in the Problem View, select *Diagram — Validate* from the menu).

The Error Log Other than the Problem View, the Error Log will log problems with the editor itself. So if you encounter strange behaviour or in case the editor should crash you can check here for the reason and send in an error report.

3.3 Additional Views for the VSDT

The following are views that have been crafted especially for the VSDT.

The BPMN-Properties View Not to be confused with the Eclipse Properties View (see above) this view displays all the Properties (a non-visual supporting type of BPMN) that lie in the scope of the currently selected element in the active editor. E.g., when selecting a Task embedded in a Sub Process, the Properties of the Task itself, the Sub Process, and the Pool holding the Sub Process will be displayed, if any. By double-clicking one of the entries, a dialog for organizing the Properties of that element will be opened.

The Pool Visibility View This view can be used to temporarily hide selected Pools from the diagram. Each entry provides a check box which can be used to toggle the Pool's visibility on and off, which can be useful when modeling diagrams with a large number of Pools.

The RSD View With this view comes a client for he *Rich Service Directory*, providing access to a number of basic Web services which can easily be reused and orchestrated when modelling a BPEL process. The several operations of each Web service registered at the Rich Service Directory are displayed in a list from which they can be selected, inspected, and imported into the diagram. When importing Web services, the corresponding Web service descriptions and input and output messages will be created automatically.

The Web Service View Similar to the RSD View, this view can be used to inspect and import Web service descriptions into the active BPMN diagram. However, other than using the RSD view, the Web Service view does not need the RSD but will directly connect to a given Web service URL.

4 Basic Tutorial

In this chapter the user will be guided through the creation of a simple Business Process Diagram, from creating the diagram file to validation and code generation.

4.1 Creating a new Business Process Diagram

1. Start Eclipse and select a location for the workspace. This is where all the projects – BPMN and others – will be stored. When starting Eclipse for the first time, a welcome screen will be shown. Read something about the features of Eclipse, if you want, then exit the screen.

2. Select *New → Project → General → Project* in the menu bar. Open the Project Explorer to see the newly created Project.

3. On the project, select *New → Other... → Easy Service Creation → BPMN Diagram* and enter a name for the File. On the last page of the wizard some of the global settings for the Business Process Diagram, such as the Title, Description and Author name, can be set. Two files will be created. The first file, `$NAME.bpmn_diagram` is holding the layout information for the diagram, while the other one, `$NAME.bpmn`, is for the pure model data.

   Note that both files are stored in XML format and can be edited with a text editor, too. However, you should do so only to fix a broken file. The diagram file can be recreated from the model file by right clicking it and choosing to initialize the diagram file.

4. By now, the diagram file should have opened automatically; otherwise open it manually by double-clicking it. It will be opened with the graphical editor. For now, ignore the model file.

4.2 Modelling a basic Business Process

1. In the graphical editor, select **Pool** from the top of the palette and move the mouse to the canvas. Push the mouse button and drag it to the lower right to create a large Pool. Enter a name for the Pool when you are prompted to.

2. Along with the Pool also a Lane will be created. To create more Lanes, select the **Lane** element from the palette and click somewhere on the Pool (be sure not to aim at the already existing Lane). According to the BPMN specification the first

Lane will be invisible (faded out in the editor). Note that the Lanes can not be resized or moved manually. They will adapt automatically to the elements within. You can right-click the Pool and select *Auto Size* from the format menu and the Pool will automatically adapt its size to the Lane's size, too.

3. Let's create some **Flow Objects** inside of the Lane. Select one of the Flow Object from the palette, i.e. Events, Activities and Gateways, and click inside of the Lane. In case you selected the Event, a small menu will appear, asking whether to create a Start, End, or Intermediate Event; otherwise the element will be created right away.

4. Select the **Sequence Flow** icon from the palette and connect the several Flow Objects by pressing the mouse button on the source and dragging it to the target. When connecting the Activity be sure to aim for the label. If you hit the Activity's compartment you can not create a connection. You can change the routing style from the toolbar or add more bendpoints to a connection by dragging it.

5. Next, we will define the process data, i.e. Properties associated to the Pool's Process. Open the Pool's overview property sheet and click *Organize Process Properties* or select the respective item from the Pool's context menu. Create some properties using the buttons in the shown dialog and edit the values of the selected Property using the text fields in the lower part of the dialog.

6. Now select the **Message Flow** icon from the palette. Select an Activity or an End Event as source and draw the Message Flow to an element in a different Pool, or to some point beneath the Pool and select to create a new Pool element there.

7. Finally, we will associate an Activity with a Data Object (however, this will not affect the generated BPEL code). Select the Data Object from the palette and create it on the canvas. Select the Association connection from the palette and connect the Data Object to the Activity. Select *BPMN → Initialize Input/Outpu Set* from the Activity's context menu, depending on the associations direction. Notice the new Input Element in the Activity's property sheet. This Input Set references all the Activity's incoming/outgoing Data Objects.

4.3 Web Service Invocation

Note: In case of the export to BPEL this will map to a Web service call, but can map to similar concepts, too, depending on the target language.

1. Right-click the Message Flow or open its property sheet and select *Initialize Message*. Note that by doing so the End Event's type changes to `Message`. A new (non-visual) Message object has been created and associated with the Pool's Participants, the Message Flow, and its source and target, if possible.

2. To define a Web service invocation, select the *Organize Implementations* and *Organize Messages* buttons from the Business Process Diagram's property sheet. Select the newly created Messages and Implementations (Web services) from the list and set the values according to the Web service to invoke. Alternatively, Web services can also be imported either using the RSD View or the Web Service View, which is much more comfortable and will be explained in depth later.

3. Just like Processes and Activities, Messages can have Properties. Again, the dialog can be accessed via the property sheet or the context menu. Add some properties to the newly created Message(s), being the input and output parameters of the respective Web service.

4. To pass the parameter values to the Message, create one or more Assignments on the Flow Object the Message is going in or out of. Open the *Organize Assignment Dialog* via the Flow Object's property sheet or context menu and create the Assignments. Select a Property to assign the value to, e.g. one of the input parameters of the Web service's input message, and enter a from expression. To refer to a Property in the expression, just type in the Property's name with a leading `$`, e.g. `$foo + 1`. Note the assign time value: if this is set to before, the assignment will be made before, the Activity is executed, i.e. the Web service is invoked, otherwise the assignment will be made afterwards. Thus this value should be set to *before*, when passing values from the process to a Web service's input parameter, and to *after*, when passing values from the Web service's output parameter back to the process.

4.4 Validation and Code Generation

1. To validate the diagram, select *Diagram* → *Validate* from the menu, or by clicking the checkmark symbol in the tool bar. You might notice some error or warning marks in your diagram or entries in the problem view. You should fix these problems before exporting the diagram to executable code and validate the diagram again.

2. Once the diagram shows no more errors, it can be exported to executable code.[1] Select *Export...* from the file menu or from the model file's context menu. Select the desired target language from the *BPMN Export* group and proceed through the dialog. Select the model file(s) to be exported (not the diagram files), adjust the target directory or the other options, if necessary, and hit the *Finish* button.

3. The export might take some seconds. If the model is sound, the output files will be created in a new directory in the specified target directory, named after the Business Process Diagram. By default, also a log file will be created along with the model files in the directory *exportLogs* in the specified target directory.

[1] However, whether the code will be right away executable also depends on whether there are no other modelling mistakes in the diagram that can not be detected by the automated validation.

4. If the process has been modelled accurately, the resulting program can be readily executable. Still it is recommended to check the result with a native editor for the respective language, to be sure the files are free from defects.

5 Selected Features

This chapter will give further insight on how to use some of the features of the Visual Service Design Tool.

5.1 GMF Modelling Assistance

The Eclipse Graphical Modelling framework the VSDT is based upon comes with a number of valuable modelling assistance features. In the following some of these will be briefly introduced. Figure 5.1 is showing the modelling assistance in use.

- When resting the mouse on top of a compartment, a small palette will show up, showing the elements that can be placed in this compartment. Thus one does not have to go all the way back to the palette for creating a new node.
- When resting the mouse on top of a node, small arrows going in and out of the node will appear. By dragging these arrows new connections can be drawn.[1]
- When a connection is drawn and the mouse button is released over the canvas or another compartment, a node can be created in that place along with the connection.
- In case multiple node- or connection-types can be created using a given tool in a given context, the user will be prompted to select one.

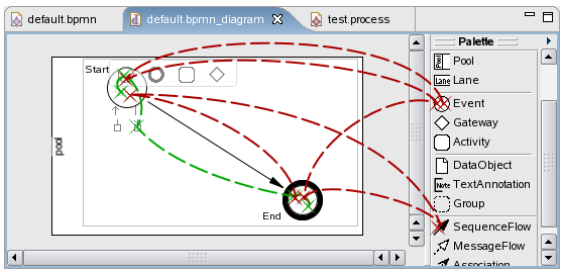

Figure 5.1: Mouse movement with and without the use of the Modelling Assistant.

[1] Depending on the location, different connections will be offered: In the top and bottom region of a node incomimng and outgoing Message Flows, in the left half incoming Sequence Flows and Assiciations, and in the right half outgoing Sequence Flows and Associations respectively.

5.2 Managing Non-Visual Elements

For each of the non-visual elements — Properties, Assignments, Messages, Implementations, and Participants — a management dialog has been written. The dialogs follow a clear and recognizable layout, showing the elements as-is in a list along with a number of buttons for inserting, removing and sorting of the elements and text fields for editing the attributes of the currenty selected item (see Figure 5.2).

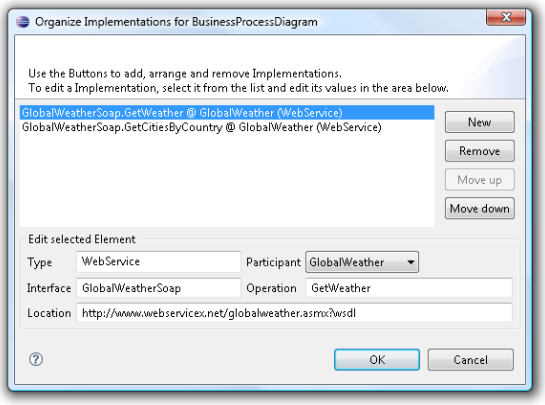

Figure 5.2: A Supporting Type Organization Dialog.

The various dialogs can be accessed in the following ways:

- The Organize Properties Dialog can be accessed via the context menu and property sheet of Pools, Activities and Message Flows, and by double-clicking an element in the BPMN-Properties View or a Message Flow.[2]
- The Organize Assignments Dialog can be accessed via the context menu and property sheet of Pools and Flow Objects, the context menu of Sequence Flows, and by double-clicking Flow Objects or Sequence Flows.[3]
- The Organize Messages Dialog can be accessed via the context menu and property sheet of the Business Process Diagram and the property sheet of Message Flows.
- The Organize Implementations Dialog can be accessed via the context menu and property sheet of the Business Process Diagram.
- The Organize Participants Dialog can be accessed via the context menu and property sheet of the Business Process Diagram.

[2] In the case of Message Flows, the Properties of the underlying Message, if any, will be edited, and in the case of Pools, the Properties of the Pool's Process.

[3] In the case of Sequence Flows, the Assignments of the respective Gate (exit from a Gateway), if any, will be edited.

5.3 Expressions

The BPMN standard does not specify an expression language to be used. Instead, it is assumed that the language of the target framework is used, e.g. XPath. However, in a tool that provides transformations to various target frameworks this is not an option. While the diagram structure could be translated to the syntax of the target system, the expression, given that they are written in an unknown language, could not – although all those languages might be very similar. To address this flaw, the VSDT comes with its own, very simple expression language, the *VSDT Expression Language*, or VXL for short. The advantage of using VXL is that it provides a greatest common divisor of the expression languages used in the target frameworks. Thus, most expressions can be given using VXL, in which case they can be validated and – more importantly – parsed and translated to the respective expression languages used in the target frameworks.

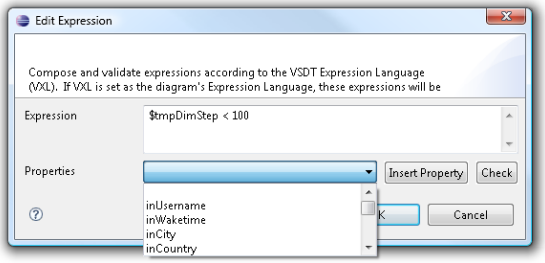

Figure 5.3: The Edit Expression Dialog.

Each text field referring to an Expression in the dialogs and property sheets of the VSDT provides a small button for opening the Edit Expression Dialog, which can be seen in Figure 5.3. This dialog not only provides a large text field for editing the Expression, but also a list of all Properties visible in the scope of the element owning the Expression, which can be selected from the list and inserted into the expression. Further, the *Check* button can be used to validate the Expression, which will check both the syntax and the availability of the variables used in the expression.

Note that there is no type checking yet. However, this feature is on the agenda, and will be imlemented as soon as possible.

5.4 VSDT Views

The following sections provide an overview about the special views introduced in the Visual Service Design Tool.

5.4.1 The BPMN-Properties View

The BPMN-Properties View (see Figure 5.4) provides an easy way to inspect the Properties in the scope of a given BPMN element, i.e. the Properties that can be used in an Assignment owned by that element. The property scope of a BPMN element comprises

the Properties of (a) that element itself, e.g. an Activity, (b) Messages going in and out of that element, e.g. the in- and output parameters of a Web service call, and (c) the (transitive) parents of that element, e.g. (Sub-) Processes.

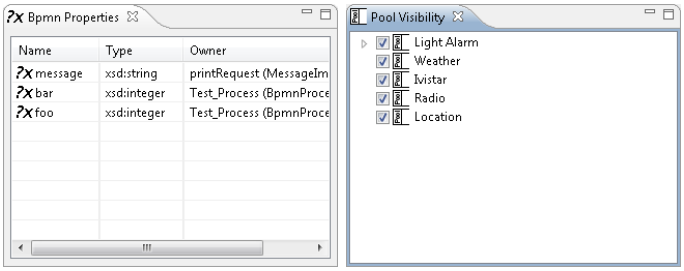

Figure 5.4: The BPMN Properties View and Pool Visibility View.

The Properties are displayed in three columns, showing the name and the type of the Property and the name and the element type of the Property's parent element. The properties an be sorted by clicking on the column heads. By double-clicking on a Property, an Organize Properties Dialog will be opened for the Property's parent element.

5.4.2 The Pool Visibility View

In the Pool Visibility View, which is also seen in Figure 5.4, all the Pools in the diagram are displayed. If the check box in front of an entry is unchecked, the corresponding Pool and all incoming and outgoing connections, e.g. Message Flows, will be hidden. This feature can be of some use in diagrams holding many Pools, e.g. when modeling three or more interconnected Pools, Message Flows going from the first to the third Pool might cross the second Pool, which can be confusing when editing that Pool. In this case, the first or the third Pool may be hidden, so the Message Flows (which are then hidden, too) do not longer obstruct the view on the second Pool. In the same way the Pools and Message Flows can be shown again by checking the corresponding check box. Note that these settings are not persisted, so when closing and re-opening a diagram all Pools will be visible again. Further, the Pool entries can be expanded, showing an outline of the elements within. However, this feature, which might some day supplement the Outline View, is still at an early stage.

5.4.3 The Web Service View

The Web Service View (see Figure 5.5) provides access to Web Services, which can be inspected and imported into the currently opened diagram. Web Services can be added to the list by clicking the Open button and entering the exact URL of the WSDL definitions file. The Web Service is displayed as a tree, including the various Messages and their types, nad the Port Types, their Operations, and their In- and Output Messages. By clicking the Info button, the complete WSDL definition is shown in plain XML. Most

importantly, Messages and Operations can be imported from the Web service description into the Business Process Diagrams, so they can be reused in a Web service invocation.

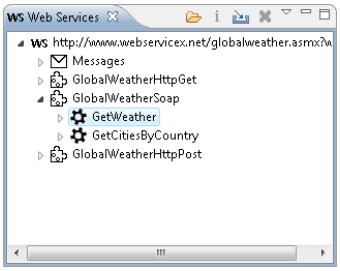

Figure 5.5: The Web Services View.

5.4.4 The RSD View

Similar to the Web Service View, the RSD view can be used to inspect and import Web services into the process diagram. However, in case of the RSD View not a single Web service is inspected, but the content of a *Rich Service Directory*, a special kind of Web service repository also capable of advertising services and operations from other sources, e.g. UPnP devices or JIAC TNG actions. However, the RSD implementation is still at an early stage, and thus by now the use of the RSD View is limited.

5.5 Inserting Elements and Patterns

By right-clicking on a Sequence Flow the *Insert...* menu can be reached. Here it is possible to insert a new element in between the source and the target of that Sequence Flow, which is very useful for extending existing diagrams. The existing sequence Flow will be reoriented to the new element, preserving existing attributes such as the condition, and a second Sequence Flow is drawn from the new element to the existing Sequence Flow's former target.

Apart from simple elements such as Activities, Intermediate Events and Gateways, it is also possible to insert complex workflow patterns, such as a split/merge block or a loop. This does not only greatly reduce the time needed for the diagram creation[4], but also ensures that the workflow is correct (Referred to as "correctness by construction").

Note that by now the layout of the diagram will not be adapted to the newly inserted elements, thus the user will have to rearrange the surrounding elements to make room for the new nodes.

[4]Reducing the pure editing time by up to 70% according to [3].

5.6 Model Validation

BPMN diagrams can be checked to conform to the constraints given in the BPMN specification by selecting *Diagram* → *Validate* from the menu or by clicking the checkmark icon in the tool bar. Afterwards errors will be listed in the problem view. Additionally, faulty elements will be marked with a respective icon in the process graph.

6 Preferences

This chapter will explain the several preferences that can be set for configuring the VSDT for your personal needs.

The preference pages can be accessed by navigating to *Window* → *Preferences...* in the menu and selecting *BPMN Diagram* from the list. The following sections will each explain one of the pages.

6.1 Basic GMF Settings

These settings are for improving performance by deactivating certain features of the GMF runtime, or for setting the default colors and fonts for certain diagram elements. As these features are self-explaining we will not go further into detail here.

6.2 Other Settings

These settings are specific for the Visual Service Design Tool as a BPMN editor:

- **Author** sets the default author for all new Business Process Diagrams created.
- **Enable Activity Icons** activates additional markers placed in the corners of Activities. These markers use intuitive symbols to indicate the Activity's type and whether the Activity has any Assignments and/or Properties.
- **Use Additional Colors** helps to distinguish the several diagram elements by the use of colors.

Note that both the Enable Activity Icons and the Use Additional Colors settings take effect only after closing and reopening a given Business Process Diagram file.

7 Model Transformation

The core of the Visual Service Design Tool clearly is the transformation to executable code. While by now the transformation to BPEL is the only one that can be conveniently used in practice, there are currently several other transformations under development.

For directions on how to invoke the transformation please refer to Section 4.4.

7.1 Understanding the Transformation Framework

This section will provide a brief introduction in the basics of the transformation framework. The transformation framework has been designed from the very beginning to be as *extensible* and *reusable* as possible. For that purpose the process of transformation has been subdivided into several stages, which are sequentially applied to the input model:

1. In the *Validation* stage, all identifiers are validated to contain only characters that are legal with respect to the given target language. Further, the validation will check if each element needed is in place and providing clear error messages in case something is missing.

2. The intent of the *Normalisation* stage is to put the process diagram in a uniform form, and to transform it to a semantically equivalent representation of the diagram following more strict constraints than those given in the BPMN specification.

3. In the *Structure Mapping* stage, the model is searched for graph patterns which are semantically equivalent to block structures. When such patterns are found, they are replaced with a special structured element, until the entire process within each Pool has been reduced to a single complex element, e.g. a sequence, or until it can not be reduced any further due to structural flaws.

4. In the *Element Mapping* stage, the several BPMN elements are mapped to their counterparts in the target language, e.g. BPEL or JIAC (Section 7.2.1 and 7.2.2).

5. In the *Clean Up* stage, a set of rules is applied on the newly created target model, improving the readability of the generated code and removing redundancies.

A simple example of the consecutive execution of normalisation and structure mapping can be seen in Figure 7.1.

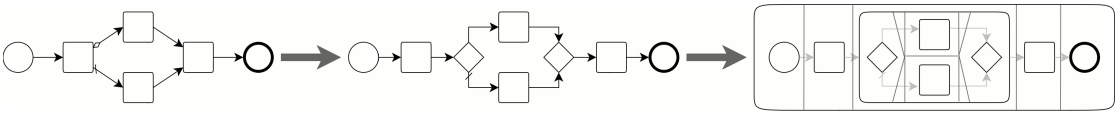

Figure 7.1: Simple example of normalisation and structure mapping.

7.1.1 Transformation of Expressions

Besides the actual workflow, Expressions that are used in assignments and conditions have to be translated, too. This can be done only if the Expression Language is set to "VSDT Expression Language" or "VXL".[1] If then the *Translate Expression* option is checked in the Export Wizard (see Figure 7.2) these expressions will be parsed and, if possible, translated to the respective target language.

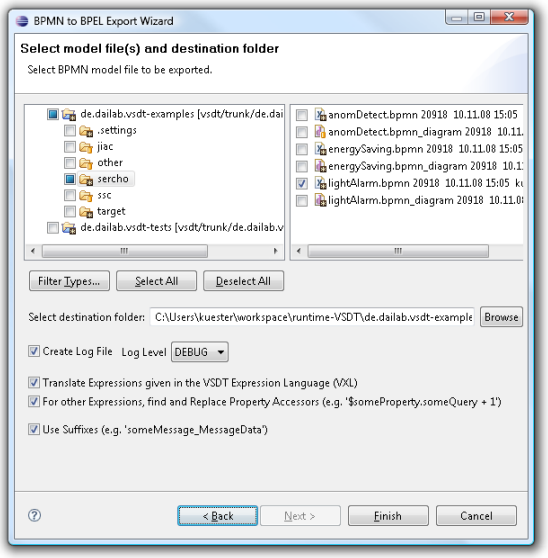

Figure 7.2: BPEL Export Wizard with Expression Translation checked.

Still there may be cases when VXL does not have enough expressive power. In this case the option can be disabled (or the Expression Language can be changed) and the *Replace Property Accessors* option can be checked. In this case, the Expressions will only be scanned for Property Accessors using VXL (e.g. `$foo.bar`) which will then be translated to the syntax of the target language. Thus, the simple VXL variables can be embedded in expressions of another language. For instance, in the case of BPEL, an expression like `$foo.bar + 1`, might be changed to `bpws:getVariableData('Proc_ProcessData','foo','bar')+1`. Thus the user does not have to care about the way Properties are aggregated to variables in the transformation to that language but

[1] The Expression Language can be set either globally in the Diagram properties or individually for each Expression.

can simply use a Property's name.

7.2 Transformation Implementations

The following sections describe in short the various transformations that have already been implemented.

7.2.1 Transformation to BPEL

The transformation to BPEL presented in this work covers nearly the entire mapping as given in the BPMN specification [6, Chapter 11], including event handlers, inclusive OR and event-based XOR Gateways, just to name a few.

Export Nevertheless there are some elements for which the mapping is not given very clearly, such as TIMER Start Events, independent Sub Processes or multi-instance parallel loops. While these elements will be transformed as described in the specification, the resulting BPEL processes will require some amount of manual refinement. Besides the BPEL process files a WSDL definitions file is created, holding the message types derived from the process properties and the input and output messages and interfaces (port types) for the several Web services being orchestrated by the process. Still, the WSDL's binding and service blocks and necessary schema types, if any, can not be generated automatically yet, due to insufficient information in the source model.

In the validation, all identifiers are tested to contain only characters that are legal with respect to BPEL, and all expressions used e.g. in Assignments and loop conditions are translated to XPath, if possible. Properties are aggregated to one variable per Process or Message, for instance, if a Process `Proc` has a Property `foo`, `foo` will be a Part of Variable `Proc_ProcessData`.

Import The Import from BPEL to BPMN is still at an early stage. While basic control flow can be imported, there are still problems e.g. with event- and fault handlers. Further one should be aware that the export to BPEL does not preserve all information in the BPMN diagram, thus diagrams re-imported after being exportet to BPEL will most likely be less readable than before, although they may be semantically equivalent.

7.2.2 Transformation to JIAC

Concerning our goal of transforming BPMN diagrams to multi-agent systems (MAS) the work is still at an early stage. First, a *normal form* for BPMN diagrams has been investigated, to facilitate the mapping [1]. Later, the first steps of the actual mapping have been developed, basically mapping Pools to agents, Processes and Flow Objects to the agents' plans and the control flow, and Message Flow to the exchange of messages between the agents [2].

Export Prototypic transformations targeting the agent frameworks JIAC IV [7] and JIAC TNG have already been implemented.[2] As the theoretical part of the mapping is not yet fully matured, there is still some work to do. However, with the given transformation framework every addition to the mapping can quickly be adopted.

7.2.3 Transformation to STP-BPMN

Since BPMN does not provide a standardized interchange format, transferring a diagram from one editor to another is difficult. For that reason the VSDT provides a Transformation to the popular Eclipse STP-BPMN Editor. Thus one can export a Diagram to STP-BPMN and carry it to a colleague using that editor, or someone formerly using the STP editor can import his existing files when changing to the VSDT.

Export The export to STP is nearly complete. The only element that are not mapped are Lanes, which is because the STP editor is encountering problems when initializing the diagram files containing Lanes. Alas, the same also applies to Embedded Subprocesses, as there are problems opening those diagrams. However, this is a problem with the STP editor that will get fixed soon.

Import The import from STP works very well, with the exception that, again, Lanes will not be mapped. *Note* that the transformation to STP will only create the BPMN model files, but not the diagram files; those files, holding the graphical information, have to be generated anew from the model file. Thus, the transformation does not preserve the layout of the diagrams.

7.3 Limitations

Although the transformation framework is quite powerful, it is important to understand that there are still some limitations, both in the mapping of structures and elements.

- Given a well-structured workflow, the transformation will yield very good results. However, as BPMN is more powerful that block-oriented languages, such as BPEL and JIAC, there will always be graph structures that can not be transformed to an equivalent program of the target language. In that case the diagram should be manually restructured *before* being exportet.
- The mapping to BPEL is as complete as possible, according to the specification. Still, there are some points in the specification that are missing, unclear or ambiguous. The transformation to BPEL is implementing the specification in almost every detail, but regarding these elements a small amount of manual refinement may be necessary.

[2] http://www.jiac.de/

- The mapping to JIAC is still at an early stage. JIAC models created with this transformation will require a large amount of manual work.

We are constantly investigating ways of extending both the quality of the element mappings and the performance of the structure mapping.

8 FAQ

This chapter will give valuable advice on how to solve several tasks, solve problems, and answer other questions related to the Visual Service Design Tool.

8.1 How to...

. . . draw Flow Objects on the Canvas? Although sometimes such diagrams can be seen, Flow Objects *can not* be drawn on the canvas. Instead, each Flow Object has to be contained in a Lane, and the Lane has to be inside of a Pool. However, according to the specification one Pool per diagram may have an invisible border, which can be set using the property sheets.

. . . create an Embedded Sub Process? First, create a basic Activity. Then, in the Activity's property sheet, set the activity type to Embedded Sub Process. Now you can resize the Activity/Sub Process to an appropriate size and create further Flow Objects inside of it.

. . . create an Intermediate Event on an Activity's boundary? For creating an Intermediate Event on an Activity's boundary, i.e. an event- or error handler, you should select the Event type from the palette and click the Activity's label or boundary, and *not* the compartment. Alternatively, you can use the miniature palette that appears when hovering the mouse over the Activity's label or boundary. Once created, the Intermediate Event can be moved freely around the Activity's border.

. . . draw Artifacts inside of a Pool? Contrary to Flow Objects, Artifacts can not be created inside of a Pool. However, you can create the Artifact on the canvas and drag it over the Pool. But remember that the Artifact is *over*, and not *in*, the Pool, so it will not be moved together with the Pool.

8.2 Frequently Asked Questions

Not so many questions asked yet.

8.3 Troubleshooting

I've drawn a legal Business Process Diagram, but when I try to export it the resulting code is broken. The diagram has to be in a certain form so it can successfully be

transformed. The diagram has to be *structured*. Primarily, there must not be blocks or loops with multiple entry or exit points. Although the VSDT can handle some forms of slight unstructuredness, for best results you should design your diagram in a way it can be mapped to a block-oriented language in a straight-forward manner.

I've created a Business Process Diagram using element XYZ, but in the resulting code this element is missing. Not each single feature of BPMN can be taken into account in all of the transformations yet. For those elements that can not be mapped, a no-operation element will be created in the target model, such as a `empty` Activity in BPEL or a `logwarn` in JIAC. Be sure to substitute these elements with a proper implementation after the transformation.

A The Business Process Modelling Notation

The *Business Process Modelling Notation* [6] was first published by the BPMI and has later been adopted by the OMG. The goal of the development of BPMN was to create a standardized modelling notation for business processes and by that to reduce the confusion created by dozens of proprietary business process notations. A brief introduction to BPMN is given for instance in [8].

There are four basic categories of element types in the notation: Flow Objects (Events, Activities and Gateways), Connecting Objects (Sequence Flows, Message Flows and Associations), Swimlanes (Pools and Lanes) and Artifacts (Data Objects, Groups and Annotations).

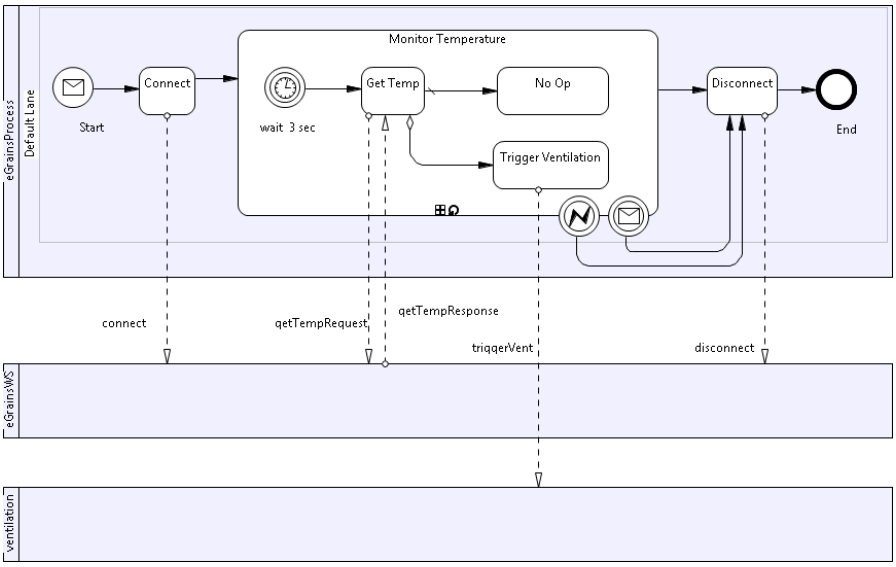

Figure A.1: Business Process Modelling Notation Example Diagram

Figure A.1 shows a simple example for an e-mail client periodically looking for new mail. The elements are quite self-descriptive and most of them are already known from other notations, so the basics of the BPMN are readily understandable for all business analysts, architects and developers and even for non-experts. At the same time BPMN provides a large variety of subtypes for each of the Flow Objects and every element type is enriched with many non-graphical attributes, making the models sufficiently detailed for being exported to executable languages while keeping the visual notation concise and

understandable.

A problem with BPMN is that it is mainly a *notation*. Although the specification describes many non-graphical attributes and a mapping to a formal language, it neither states an exchange format, like an XSD, nor clear semantics for all of the elements. Still the Business Process Modelling Notation can be used throughout the whole software engineering lifecycle, from a simplified model at the requirements analysis up to a highly detailed model that can be used for generating code for an executable language.

A.1 BPMN Elements

This section is intended to give a brief introduction of each of the basic element groups:

- **Flow Objects**: Events (Start, End and Intermediate), Activities (Task and Sub-process) and Gateways
- **Connecting Objects**: Sequence Flows, Message Flows and Associations
- **Swimlanes**: Pools and Lanes
- **Artifacts**: Data Objects, Groups and Annotations

A.1.1 Flow Objects

The category of *Flow Objects*, the most important elements in BPMN, is made up of *Events*, *Activities* and *Gateways*. All Flow Objects are held in Lanes (see below).

Events are things that *happen*, like a message arriving, an alarm, or an error, and often they mark the beginning and the end of the process. The graphical notation is a circle. They are subdivided into *Start Events*, *Intermediate Events* and *End Events*, which will determine the circle's border (see figure A.2).

Figure A.2: BPMN Event types. From left to right: Start Event, Intermediate Event, End Event

Further all three Event types have a variety of subtypes which will determine e.g. a Start Event's *trigger* and an end Event's *result*. Each of these subtypes can be distinguished by a different icon in the center of the Event figure (see figure A.3) and results in a number of attributes to be set for the Event.

Basically, an **Activity** is something that is *done*. Activities subdivide in *Tasks*, which are atomic Activities, and *Sub Processes*, which are composite Activities. The graphical notation for an Activity is a rounded rectangle with the Activity's name inside of it. Sub Processes are marked with a small ⊞ sign on the bottom line (see figure A.4).

Figure A.3: BPMN Event sub types. From left to right: Rule, Timer, Message, Link, Multiple, Cancel, Compensate, Error, Terminate

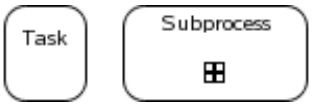

Figure A.4: BPMN Activity types. From left to right: Task, Subprocess

Like Events, Activities also have some specializations, each one with special attributes: They can be for instance a *Send* or a *Receive* Task, stand for some *Manual* work to be done, execute a *Script* or, in case of the *Independent* Sub Process, represent a whole business process, just to name a few. All of these subtypes have the same graphical representation, but modellers and modelling tools are free to extend the diagrams with additional markers for the subtypes.

What makes Activities stand out from the other Flow Objects is that they can *loop*. Although in BPMN loops also can be defined by simply connecting a Sequence Flow to an upstream Flow Object, which might be easier to understand by non-experts, it's seen as better style to use looping Sub Processes. A looping Activity is marked with a small counter-clockwise arrow on its bottom line.

Gateways provide wide capabilities in modelling all kinds of splitting and merging behavior. Figure A.5 shown the different kinds of Gateways. Depending on whether the Gateway has multiple incoming or outgoing Sequence Flows – or even both – it has different semantics, like forking and/or joining the flows. However, Gateways are not the only way for modelling forking and joining of flows. In some cases the same semantics can be reached by omitting the Gateway and connecting multiple Sequence Flows directly to an Activity[1].

Figure A.5: BPMN Gateway types. From left to right: Data based XOR (with and without marker), Event based XOR, Inclusive OR, Complex, AND

[1] this is not allowed for Events

A.1.2 Connecting Objects

The most important connections are *Sequence Flows* and *Message Flows*. Sequence Flows represent the flow of control and connect Flow Objects within a Pool in the order of execution. Message Flow represents messages – not necessarily data – being exchanged exclusively between Pools. See figure A.6 for the connections' graphical notation.

Figure A.6: BPMN Connection types. From left to right: Sequence Flow, Message Flow, Association

The third connection, the *Association*, is mainly used for documentation reason, for instance to connect a Text Annotation to a Flow Object that needs further explanation. Still there is an exceptions to this rule: For connecting a compensating Activity to a compensation Event an Association is used instead of a Sequence Flow. An Association's arrow heads are optional.

A.1.3 Swimlanes

Swimlanes can be *Pools* and *Lanes*. Each Pool represents one Participant in the business process, while Lanes are used to partition a Pool. Doing so each of a company's departments could be represented by a Lane while the Pool stands for the company itself. Typically Pools and Lanes are oriented horizontally, but they may be oriented vertically, too. Further, Lanes may cross or contain other Lanes, but those techniques are poorly documented and usually not used. Figure A.7 shows a small horizontally oriented Pool containing two empty Lanes.

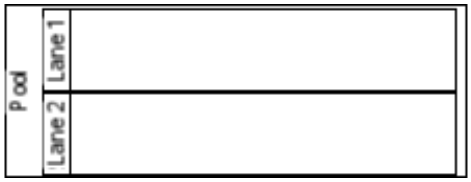

Figure A.7: BPMN Swimlanes. A Pool with two Lanes

A.1.4 Artifacts

The main purpose of *Artifacts* is documentation. However, like Associations, in some situations they can have semantics, e.g. when a *Data Object* is referenced by an Activity as input. Data Objects represent everything that can be input or output of some Activity.

In most cases this will be a file, but since Activities can be *Manual Tasks*, too, a Data Object could also stand for something physical.

The other two Artifacts, *Group* and *Text Annotation*, are solely used for documentation. See figure A.8 for their graphical notation.

Figure A.8: BPMN Artifacts. From left to right: Data Object, Group, Text Annotation

The specification states that this category may be extended by proprietary artifacts which could have more semantics, too. This way BPMN can be extended with new elements to represent concepts that were not considered in the original specification.

A.2 Levels of Complexity

BPMN can be seen as having at least three levels of complexity.

1. Basic Types: All diagrams are made up of the basic elements of the four categories: Events, Activities, Gateways, Connections, Pools and Artifacts. These can be understood easily even by non-experts.

2. Subtypes: The Flow Objects each have several subtypes, e.g. Timer Events, Receive Tasks and Inclusive Gateways. Using the same shapes as the basic elements enriched with some additional graphical information, like an icon, the symbol's basic type can be clearly identified by non-experts while providing additional visual information for the professionals.

3. Attributes: Each of the BPMN element types provides a large number of both primitive and complex attributes. While some of these attributes are visible in the diagram, like a flow object's name or subtype, most are non-graphical. These attributes enrich the diagram with the formal semantics necessary for the export to an executable language while not polluting the visual notation with too many details.

If the process shall be mapped to a executable language the third level is very important: Not only does it give values for many attributes that otherwise would have to be set manually. Some of the visual elements of the BPMN do not have semantics "on their own". Message Flows for example do *not* have a mapping to WS-BPEL. Instead the source Activity has to be of type `Send` and the target Activity of type `Receive`, and both have to reference the same non-graphical `Message` element. This is necessary in cases when the communications partner is not in the same diagram and thus a Message Flow can not be drawn.

Of course it is free to the designers of new mappings to map the Message Flow, if it is available, without insisting on the existence of the non-graphical Message element.

A.3 Export and Code Generation

One of the main purposes of the Business Process Modelling Notation is to provide a graphical notation that can be used to generate executable code from it.

A mapping to WS-BPEL is given in the BPMN Specification. As a matter of fact, BPMN has been tailored for the mapping to WS-BPEL, which can be seen in many attributes which are needed only for the mapping. Most of these attributes can be reused for mappings to other languages, too, e.g. such common concepts as properties and assignments.

On the other hand BPMN has more expressive power than BPEL. A diagram in BPMN is a directed graph, while BPEL, and in fact most other executive languages as well, are block oriented, making the export to a semantically equivalent program complicated and in some cases impossible. Numerous papers have been written on how to identify block structures within a BPMN diagram or how to alter an existing diagram to conform to block structure. However, not every diagram can be refactored like that.

While the basic elements such as Flow Objects should neither be altered nor extended by new elements the BPMN Specification encourages the introduction of new, domain-specific Artifacts to be used in mappings to executable languages other than BPEL. These elements can be associated with the original BPMN elements and represent concepts that were not considered in the original BPMN specification.

B The VSDT Expression Language (VXL)

The BPMN standard does not specify an expression language to be used. Instead, it is assumed that the language of the target framework is used, e.g. XPath. However, in a tool that provides transformations to various target frameworks this is not an option. While the diagram structure could be translated to the syntax of the target system, the expression, given that they are written in an unknown language, could not – although all those languages might be very similar. To address this flaw, the VSDT comes with its own, very simple expression language, the *VSDT Expression Language*, or VXL for short.

B.1 Language Features and Syntax

The VSDT Expression language has been designed to be the greatest common divisor of the expression languages used in the target frameworks. Thus, most expressions can be given using VXL, in which case they can be validated and – more importantly – parsed and translated to the respective expression languages used in the target frameworks.

Below, the complete syntax of VXL is given. As can be seen, it is not much different from that of other languages. Variables have to be marked with a leading `$`, followed by the name of a Property in the scope of the owner of the Expression. The Variable may be followed by one or more accessors, e.g. for access to an array element (e.g. `$foo[2]`) or a field (e.g. `$foo.bar`), or combinations thereof (e.g. `$foo[$n+1].bar`); of course accessors can only be used if the target language and data type supports them. An explanation of the various operations and comparisions can be found in Table B.1.

```
Term:           Head (Tail)?;
Head:           BracketTerm | Negation | Atom;
Tail:           Operator Term;
BracketTerm:    "(" Term ")";
Negation:       "!" Term;
Atom:           Value | Variable;
Variable:       "$" ID (Accessor)?;
Accessor:       ArrayAccessor | FieldAccessor;
ArrayAccessor:  "[" Term "]" (Accessor)?;
FieldAccessor:  "." ID (Accessor)?;
Value:          STRING | INT | FLOAT| "true" | "false" | "null";
Operator :      "<" | "<=" | "==" | "!=" | ">" | ">=" |
                "+" | "-" | "*" | "/" | "%" | "&&" | "||" | "++";
```

Table B.1: VXL Operations and Comparisions

	Name	**Symbol**
Operations	Addition	`+`
	Substraction	`-`
	Multiplication	`*`
	Division	`/`
	Modulo	`%`
	Concatenation	`++`
	Logical AND	`&&`
	Logical OR	`||`
Comparisions	Equal	`==`
	Not Equal	`!=`
	Lesser	`<`
	Lesser or Equal	`<=`
	Greater	`>`
	Greater or Equal	`>=`

Bibliography

[1] Holger Endert, Benjamin Hirsch, Tobias Küster, and Sahin Albayrak. Towards a mapping from BPMN to agents. In Jingshan Huang, Ryszard Kowalczyk, Zakaria Maamar, David Martin, Ingo Müller, Suzette Stoutenburg, and Katia P. Sycara, editors, *Service-Oriented Computing: Agents, Semantics, and Engineering*, volume 4505 of *LNCS*, pages 92–106. Springer Berlin / Heidelberg, 2007.

[2] Holger Endert, Tobias Küster, Benjamin Hirsch, and Sahin Albayrak. Mapping BPMN to agents: An analysis. In Matteo Baldoni, Cristina Baroglio, and Viviana Mascardi, editors, *Agents, Web-Services, and Ontologies Integrated Methodologies*, pages 43–58, 2007.

[3] Thomas Gschwind, Jana Koehler, and Janette Wong. Applying patterns during business process modeling. In *Business Process Management*, pages 4–19, 2008.

[4] Tobias Küster. Development of a visual service design tool providing a mapping from BPMN to JIAC. Diploma thesis, Technische Universität Berlin, April 2007.

[5] Tobias Küster and Axel Heßler. Towards transformations from BPMN to heterogeneous systems. In Massima Mecella and Jian Yang, editors, *BPM2008 Workshop Proceedings*, 2008.

[6] Object Management Group. Business Process Modeling Notation (BPMN) Specification. Final Adopted Specification dtc/06-02-01, OMG, 2006. `http://www.bpmn.org/Documents/OMGFinalAdoptedBPMN1-0Spec06-02-01.pdf`.

[7] Ralf Sesseler. *Eine modulare Architektur für dienstbasierte Interaktionen zwischen Agenten*. PhD thesis, Technische Universität Berlin, 2002.

[8] Stephen A. White. Introduction to BPMN. Technical report, IBM Corporation, May 2004. `http://bpmn.org/Documents/Introduction%20to%20BPMN.pdf`.