VML Usability for Modelling TUI Scenarios - A Comparative Study

Eric Tobias
Service Science & Innovation Department
Public Research Centre Henri Tudor
J.F. Kennedy av. 29, L-1855 Luxembourg, Luxembourg

Eric Ras
Service Science & Innovation Department
Public Research Centre Henri Tudor
J.F. Kennedy av. 29, L-1855 Luxembourg, Luxembourg

Nuno Amálio
Laboratory for Advanced Software Systems
University of Luxembourg, Luxembourg
6, rue R. Coudenhove-Kalergi
L-1359 Luxembourg

TR-LASSY-12-06

Issued: April 18, 2012
Latest review: July 11, 2012

Abstract

Visual modelling languages have been around for quite a while. General-purpose visual modelling languages such as UML have long since become accepted and widely used in industry. They use diagrammatic notations to express a meaning that would only be hard to grasp with text. It is with the recent focus of interest in Model Driven Engineering that visual modelling languages have come into focus again. Often it is preferable to have a visual representation of a domain since it enable a broader audience to participate in and understand the modelling effort.

This report studies the suitability of general-purpose modelling languages for modelling Tangible User Interface applications. A concrete scenario using Business Process Model Notation 2 is developed and used to compare the modelling process, effort and outcome of the general-purpose visual Modelling languages. This report details all processes and gives a conclusion as to what degree the chosen visual modelling languages are suitable for the task. The most suitable modelling language will be used in the scope of a master's thesis to build a general model of a Tangible User Interface application.

Contents

1 Introduction **1**
1.1 Background . . . 1
1.2 Goals . . . 2

2 Identifying suitable languages **3**
2.1 Preliminary language criteria . . . 3
2.1.1 Visual syntax . . . 4
2.1.2 Modelling != Programming . . . 4
2.1.3 Generalisation . . . 4
2.2 Data Collection . . . 5
2.2.1 Searching . . . 5
2.2.2 Collecting . . . 5
2.3 Pre-selection . . . 6
2.4 Refining selection criteria . . . 7
2.5 Candidates . . . 7
2.5.1 UML + VOCL . . . 8
2.5.2 VCL . . . 8
2.5.3 Constraint Diagrams . . . 9

3 Building a scenario **10**
3.1 BPMN2 . . . 10
3.1.1 Simple scenario . . . 11
3.2 Preliminary TUI model . . . 11
3.2.1 A simple TUI instance . . . 12

4 Case study **13**
4.1 Measurements . . . 13
4.2 Study plan . . . 14
4.2.1 Create Start Event . . . 15
4.2.2 Create Activity . . . 15
4.2.3 Check portfolio . . . 15
4.2.4 Take action . . . 15
4.2.5 Wait for response . . . 15
4.2.6 Review portfolio . . . 16
4.2.7 Create Gateway / Create Event . . . 16
4.2.8 Link components . . . 16
4.3 Scope reduction . . . 16
4.4 Product . . . 17

4.4.1 UML + VOCL 17
4.4.2 VCL 19
4.4.3 Constraint Diagrams 22

5 Evaluation 24
5.1 Tool support 24
5.1.1 Availability 25
5.1.2 Maintainability 25
5.1.3 Latest version 25
5.1.4 Branch 25
5.2 Semantics & Transformation 26
5.2.1 Formally defined 26
5.2.2 Transformability 26
5.3 Expressivity 27
5.3.1 # X (# X) 27
5.3.2 # X satisfied (# Sat_X) 27
5.3.3 # requirements partially satisfied (# Part_Sat) 27
5.3.4 # unsatisfied requirements (# UnSat) 27
5.3.5 Ratio 28
5.4 Usability 28
5.4.1 Naming conventions 28
5.4.2 Naming fit 28
5.4.3 Documentation 29
5.4.4 Tutorial 29
5.4.5 Hands-on tutorial 29
5.4.6 Primitive mutability 29
5.4.7 Live suggestions 30
5.5 Error Checking 30
5.5.1 Time 30
5.5.2 Syntax highlighting 30
5.5.3 Degree 31
5.5.4 Error correction suggestion 31
5.5.5 Debugging possible 31
5.6 Validation & Verification 31
5.6.1 Modularity 32
5.6.2 Verification scheme 32
5.7 Weighting scheme 33
5.8 Results 33

6 Conclusion 36
6.1 Future work 37
6.2 Long time goals 37

Bibliography 40

Glossary 41

A List of literature resources 43

B Requirements **46**
B.1 BPMN2 model requirements 46
B.1.1 Structural requirements 46
B.1.2 Behavioural requirements 47
B.1.3 Constraints 47
B.2 Stock Management model requirements 47
B.2.1 Structural requirements 47
B.2.2 Behavioural requirements 47
B.2.3 Constraints 47
B.3 Tangible User Interface model requirements 48
B.3.1 Structural requirements 48
B.3.2 Behavioural requirements 48
B.3.3 Constraints 48
B.4 Tangible User Interface scenario requirements 48
B.4.1 Structural requirements 48
B.4.2 Behavioural requirements 48
B.4.3 Constraints 49

C **51**

D **52**

E Weighting evaluation criteria **53**

Chapter 1

Introduction

Tangible User Interfaces (TUIs) offer an interesting opportunity to move modelling tasks from a single-user environment into a collaborative environment [1]. Modelling cooperatively enables users with different backgrounds to participate on the same modelling effort, which helps to involve different stakeholders, facilitate feedback from users and experts and mitigate corrective barriers. A TUI for modelling will make use of common metaphors and well known principles to enrich the modelling experience and facilitate cooperation and collaboration.

1.1 Background

A desirable property of a model is that it must convey the same, unambiguous, meaning to every reader. Visual Modelling Languages (VMLs) must, therefore, be able to produce such models. A common and misleading practice is to be to invent a new domain specific language whenever it is needed rather than trying to find a suitable general-purpose language [2]. Not only will it save time but it will also enable the modeller to rely on an established language that has the backing of the community and resources to draw upon. Moreover, VMLs that have gone through a lengthy design process and several iterations may have all kinds of added bonuses such as being formally defined, offering tool support or tutorials. Due to these reason it was determined that for the purpose of modelling a TUI framework and its instance used to solve a simple BPMN2 [3] modelling scenario, an established VML was to be chosen.

1.2 Goals

The goal of this comparative study is to identify a suitable VML for modelling a TUI modelling application for BPMN2. In order to do so, the report identifies a number of suited candidates according to a number of preliminary elimination criteria. These languages are then used to model a BPMN2 cenario, and then compared based on a number of previously fixed, weighted criteria. The research question of interest is, therefore;

> *Which VML is most suitable for modelling a TUI application for building BPMN2 models?*

This requires modelling a TUI instance and the corresponding model. All criteria and scenarios will be detailed for the reader in the following sections. While this study somewhat follows a quantitative-experimental approach, the reader should take note that all choices are subjective and while the author tried to make as objective choices as possible, it is by the limited scope and time not possible to conduct a proper objective study.

Chapter 2

Identifying suitable languages

Before delving into any particular language choice, a suitable basket of languages must first be collected. The scientific community is buzzing with ideas and some lead to the creation of new languages. Sadly, this means a rather large amount of languages are out there and not all of them are visual and even less of them are not domain specific. Therefore, all unsuitable languages should be discarded before even entertaining the idea to embark on a study that is on a schedule.

The following sections will detail by what criteria the first batch of languages has been preprocessed and give more detail reasons as to why this sorting has taken place. The second section will list the methodology for finding languages and the subsequent section will educate on what languages have been eliminated and stated the reason for the elimination. The second to last section will see the selection criteria strengthened and give reasons as to why this was necessary. The last section will give a rundown of the remaining languages by the means of a small introduction and background information.

2.1 Preliminary language criteria

The criteria dressed here serve only to limit the number of languages taken into considerations. With the sheer number of languages and dialects produced by the scientific community in the domain of Software Engineering it would be nigh impossible to undertake a comparative study as a preliminary investigative step in the scope of a master's thesis. These criteria are by no means sorting languages in a qualitative fashion nor should they be interpreted as a sign of suitability other than their validity for the scope of the accompanying Master's Thesis.

2.1.1 Visual syntax

The main delimiting criteria is the requirement on languages to have a visual representation. This strong excluding criteria is inherited directly from the scope of the thesis. One of the reasons behind selecting and investigating only visual languages is the tease of using TUIs to dress the models for future TUI applications. Obviously, languages must be visual in order to be represented by a visual modelling interface without the need of interpretation or a translation to map the language into the visual domain.

One of the main selling points for TUI is the ability to work cooperatively [1]. In order to benefit from a collaboration in a business modelling scenario, bringing the technically non-versed customer together with business analysts and experts is paramount. Hence, the vertical cut in the user base in the cooperative design scenario suggest using visual paradigms such as TUI. Providing a modelling language capable of representing visual representations and metaphors used in TUI is therefore important. Staying close to and integrating the customer or at least his feedback into the design process is desired as it is thought to prevent problems during product validation and increase the quality, in regard to customer expectation, of the final product. It is one of the reasons why recent Software Engineering and Project Management methodologies such as for example Agile, have integrated ongoing customer feedback into their core teachings [4].

2.1.2 Modelling != Programming

Another strong criteria directly inherited from the thesis description is the need for the language under investigation to be a modelling language and not a programming language. Should any language fall into both categories it will be retained. Other than not fitting inside the scope of the thesis, it would make little sense to compare languages catering to different disciplines.

2.1.3 Generalisation

The goal as stated in 1.2 is to use one of the chosen languages to model a BPMN2 scenario using TUIs. While this is very domain specific to business processes and TUIs, it is highly unlikely that there is any one domain specific language suited to model both domains at once. Moreover, while this study only uses a business scenario based on a formally defined specification, future modelling processes might take a more liberal approach and need a more flexible, general-purpose modelling language.

Choosing a general-purpose modelling language also offers the benefit that it will likely not be overly specialised and catered to one particular application domain. This enables the language to use metaphors and abstractions to draw upon existing language schemata and improve language learning process [5]. As noted by Schema Theory [6], the ability to draw upon existing schemata will improve the ability to understand the new domain by reusing familiar concepts in already existing schemata. The more specialised and tailored metaphors used in domain specific languages, this benefit might be lost.

2.2 Data Collection

This section will state the methodology used to collect all relevant literature on, sometimes more, sometimes less, visual modelling languages. the first subsection will declare what services have been used and what parameters were used to search for pertinent data. The second section will enumerate the findings and further processing.

2.2.1 Searching

In order to accumulate the resources needed to make well founded choices not skewed in favour of any technology or language, the search was done online using Google Scholar [7] and Findit.lu [8] to span industrial and academic domains and gather a wide range of papers and articles. To find proprietary technologies, Google [9] was used as a search engine to scour the internet. Offline resources were not considered as they were deemed to not be up to date in regard to current technologies and standards and therefore not be suited for this study.

To gather a wide-range of literature, search terms were used in a variety of combinations. Early searches were undertaken using terms such as; `"visual modelling language" AND "visual modeling language"`, `"visual" AND ("modelling language" OR "modeling language")`, `visual AND (modeling OR modelling) AND language`. These accounted for American and British use of the different search terms. These searches were then refined with terms such as `behaviour`, `structural`, `quality OR constraint` and `requirement`. UML dominated most searched and therefore it was decided to also conduct searches using the added parameter `-UML` in order to identify UML unrelated VMLs.

2.2.2 Collecting

After several iterations and combinations of the abovementioned search criteria and an exploration of references, more than two dozen relevant resources were gathered. For a full list, please refer to the Appendix A. These resources fall into one of four broad categories. The first category is composed of papers and other resources describing modelling or simulation environments such as for example AnyLogic [10] or VENSIM [11]. The second category features mostly studies on visual notations, their application and use, as well as on emerging visual paradigms. A few examples of such resources are *"Analysing the Cognitive Effectiveness of the UCM Visual Notation"* [12] or *"Towards Symbolic Analysis of Visual Modeling Languages"* [13]. A separate category features on enhancing existing languages with a visual notation by proposing extensions or changes such as for example *"The semantics of augmented constraint diagrams"* [14]. The last category is composed of novel visual languages being described on their own such as for example UML [15] or Constraint Diagrams [16].

2.3 Pre-selection

In order to be able to make a final choice, the number of candidate languages had to be reduced. The following paragraphs explain what resources and thereby languages were discarded and for what reasons. Please note that the order was chosen by what subjectively seemed to eliminate the most resources or seemed the most obvious. The main criteria used for elimination were detailed in 2.1 and its sub-sections.

The first few resources to be discarded were all detailing simulation and modelling environments that were not providing any syntax outside of their environment. The affected items were related to AnyLogic [10], VENSIM [11], SIMILE [17] and subTextile [18]. The latter offers a very interesting approach to designing interactive systems. This could probably be used for designing widgets but the specificity of the visual programming language and hardware platform that is subTextile will not enable it to model many different domains. Moreover it focuses strongly on behaviour, neglecting structural elements. VENSIM is a simulator that allows for the modelling and simulation of business, scientific, environmental, and social systems with a focus on system dynamics and interactions. AnyLogic is more powerful but covers the same niches. Both simulators allow to model complex behaviour but are limited to the domain of system dynamics. By modelling and thereby preparing the system to base simulations on, the focus of the semantics lie more on defining agents, behaviour and interactions. Each simulator has a pragmatic approach to define the semantics which are not very formal or usable without the tool.

The unnamed language that was presented in [19] is designed to support sketch based design in the field of interface design. While the language could be ported to similar disciplines such as storyboarding or design in general, the applicability of the language to a broader field is questionable. The language was deemed to domain specific and discarded. PROGRESS [20] is a visual programming language using graph rewriting systems. It is however not a modelling language and has therefore been removed from the set of considered languages. The same reason holds for SPARCL [21], a visual logic programming language based on set partitioning constraints, and Forms/3 [22], a first-order visual language to explore the boundaries of the spreadsheet paradigm. The approach taken in [23] describing a novel visual programming language to draw and execute flowcharts aims at shifting the focus from code generation to algorithmic conception of programs. Not providing a general-purpose modelling approach, it was dropped from the set of considered languages. Due to the domain specificity of [24] it did not fall into the scope of the thesis. The Regatta approach as described in [25] is similar in semantics to BPMN2. While the approach could be broadened to include more domains, the specific aim of the approach is Business Process Engineering.

A last step of the preliminary sorting before tightening the criteria is to remove all stepping-stone-papers that had been used to uncover more languages but describe by themselves no language or language extension.[26, 27, 28, 13, 12]

2.4 Refining selection criteria

In order for the language of choice to be able to model the high level, abstract, view of the customer and subsequently be refined into ever more elaborate processes until the complete system had been modelled, the language would need to be able to represent all structural and behavioural requirements as well as be able to express constraints on those structures and behaviours. Structural requirements impose a structure onto the system. They describe what entities are considered being part of the system and how their internal structure looks like. Furthermore, associations and dependencies between these entities may be expressed. Behavioural requirements express how the system must behave. They state how it interacts with its environment and how it reacts to stimuli, what outputs it produces under which inputs. Requirements expressed as constraints may include functional, non-functional and behavioural aspects. They aim to express the boundaries and limitations of the system and the previously mentioned behavioural and structural requirements. Hence the requirement that the language of choice must be able to express behaviour on a structure given some constraints. Ideally, all of these requirements and constraints would need to be represented visually as a formal textual representation might prove a significant entry barrier for novel users.

The paper *"Visual modeling of OWL DL ontologies using UML"* [29] introduces a visual, UML-based notation for representing OWL ontologies. Ontologies inherently only express domain concepts and how they are related. The same limitation is faced by Object-Role Modelling [30] which focuses on structural concepts and their relation. Hence, behaviour and complicated constraints are not taken into consideration by these notations. A paper on visual constraint programming [31] has similar issues. The focus lies on constraints and the other requirements are neglected. Moreover, this paper takes on a programming rather than a modelling approach. The remaining thirteen papers and resources can neatly be arranged into belonging to one of three major language groups. The next section will deal with distilling a single language or dialect from each of those three categories.

2.5 Candidates

Papers from three main languages and language extension efforts remain after applying the exclusion and selection criteria. The first one revolves around UML [15]. This is not surprising considering the monolith that is UML in Software Engineering when it comes to modelling. UML as such offer many advantages but it also has some drawbacks which will be detailed in a following section. UML as such is not suited for the study. In order to express constraints, UML relies on OCL [32], the Object Constraint Language which is a precise text-based language. This would violate the strengthened criteria. The lack of a visual representation for OCL has been recognised and addressed by the Visual Object Constraint Language [33, 34], result of the efforts to generate a visualisation for OCL [35].

Another approach is highlighted in *"Executable Visual Contracts"* [36] which extends UML's graphical notation to include certain types of constraints. Since the constraints are limited in complexity by the visual representation, UML in conjunction with VOCL will be one of the languages under investigation. The final two picks, VCL [37, 38, 39, 40] and Constraint Diagrams [41, 42, 43, 16], are the sole topic in the related paper groupings. The following sections will give some details about the language selections and highlight some features.

2.5.1 UML + VOCL

The Unified Modeling Language [15], currently at version 2.4.1[3], is a well established standard which has a vast user base in the software industry. The language is generic enough for the goals of this study while still offering all the tools needed to model the structure and behaviour in the domain. However, in order to satisfy requirements on correctness and quality we need constraints. This is not possible in pure UML and requires the use of an extension to UML, the Object Constraint Language, OCL [32]. OCL is a formal declarative text language and does not offer a visual representation. To that end a Visualisation of OCL or simply Visual OCL (VOCL) will be used.

VOCL[33, 35] enables the user to express constraints in the same diagrammatic notation used to express the remainder of the UML model in. The objective of the visualisation is to further the use of a formal language to express constraints as it is thought that the formal and mathematical nature of OCL is the reason for the lack of usage [34]. Using UML to express structural and behavioural requirements while using VOCL to express quality requirements and invariants is suited for modelling the scenarios presented during the study.

2.5.2 VCL

The Visual Constraint Language (VCL) [37, 38, 39, 40] is a novel language that builds on set theory. The language is inspired by UML class diagrams who were in turn influenced by Entity-Relationship diagrams. VCL does take a step further in that it adds assertions into the structure to increase expressiveness. VCL provides a mean to not only separate concerns using modular design but also to address the different requirements separately, much like UML. Structural requirements are modelled separately from behavioural requirements which enables modellers to focus on one concern at a time. Instead of using multiple diagram types to specify behaviour as applied in UML, VCL uses a self-defined and separate kind of behaviour diagrams which incorporate contracts to completely model behaviour. Global and local quality and correctness requirements are addressed by so called assertion diagrams which express invariants.

[3]http://www.omg.org/spec/UML/2.4.1/

2.5.3 Constraint Diagrams

Transmitting knowledge through diagrammatic notation is an important step in making a domain accessible by outsiders. To this goal, visual languages and notations have been developed for quite a while. Some pioneers like Euler (18$^{\text{th}}$ century) and Venn (19$^{\text{th}}$ century) recognised the need for visualisation and introduced diagrammatic notations for mathematical and logical constructs. Constraint Diagrams is a notation that is very close to those early notations and provides a diagrammatic notation for expressing constraints akin to first order predicate logic [42]. While the notation could operate on its own, it is proposed to be used in conjunction with UML and replace the cumbersome non-visual OCL [16].

For the purpose of this study, the plain Constraint Diagrams as proposed by [16] is not sufficient. The initial purpose of Constraint Diagrams was to serve as an alternative OCL, hence the previously proposed conjunct use with UML. An augmented form of Constraint Diagrams is proposed in [42]. The extension to the language allows for Constraint Diagrams to specify structural constraints more conveniently and even define a scheme to extent existing structures. Behaviour is proposed to be expressed in a precondition-postcondition fashion, the same principle used in the Design by Contract paradigm.

Chapter 3

Building a scenario

A model needs to be useful. A model drafted just to prove a concept might be useful under certain circumstances but for the purpose of this study a different approach was chosen. When defining the scope of the thesis, BPMN2 was to be the driver of the scenario due to its recent gain in popularity in the business modelling community. Just modelling the BPMN2 specification however was out of the question. To see see if the language of choice was usable in an everyday usage scenario, it was decided to draft a small but somewhat realistic scenario to work with. This scenario is then completed with a sample TUI instance created to specifically handle the scenario in question in order to be able to express and capture the interactions between the two models.

The following sections will first introduce BPMN2 and give some pointers to find more in depth documentation. Then the BPMN2 scenario is described as well as its different levels of refinement. In the next section the preliminary TUI model will be described. The final paragraphs will give a brief introduction as to how widgets are constructed and what widgets were chosen to manipulate the BPMN2 scenario models.

3.1 BPMN2

The Business Process Model and Notation was developed by the Object Management Group (OMG) and is currently in its second version[3]. The specification introduces the notation perfectly for the context in which it will be used;

> The primary goal of **BPMN** is to provide a notation that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes. Thus, **BPMN** creates a standardized bridge for the gap between the business process design and process implementation. [3]

For dressing the scenario for the study, the targeted BPMN2 model should focus on visual concepts of the language rather than meeting the conformance criteria stated in [3]. Focussing on the visually perceived model entities reduces the complexity of the model and allows it to be used in the scope of the scenario. From the BPMN2 model an instance was generated that should serve as the cornerstone for the scenario. It was built upon a simple, stock themed, fictive, client-service provider model.

3.1.1 Simple scenario

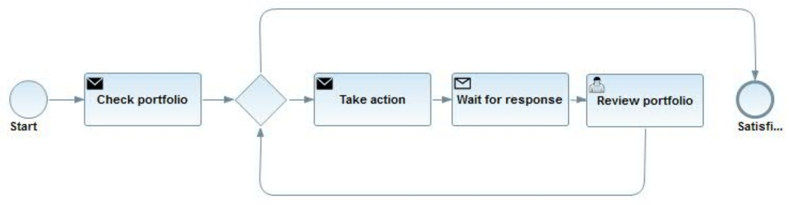


Figure 3.1: BPMN2 scenario instance at first level of refinement.

The figure above shows the basic, first level scenario which only holds one process from the customer's view. The scenario sees the customer check his portfolio and then either take action or end the scenario. After taking an action he will have to wait for a response before a final review of his portfolio. He can then choose to take another action or end the scenario. This simple introductory scenario covered a few basic aspects of BPMN2 and allowed to test some basic TUI-based modelling approaches.

The scenario was then refined another three times to produce more finely grained scenarios covering as many concepts of BPMN2 as possible. It turned out that after starting the study, that time was a rare commodity with getting aquainted to all langauges taking up quite some time. Therefore it was decided to first model the first level of the scenario to get some comparable results. As it turned out, the scenario did indeed provide enough information to make a final choice. The document detailing all scenarios, *"Capital Stock Management scenario description document"*, can be found at; `http://tinyurl.com/bqpqdj2`.

3.2 Preliminary TUI model

The preliminary TUI mode was drafted in cooperation with Paul Bicheler who is also following an internship at the Public Research Centre Henri Tudor. As of now there is no coherent model or understanding on the building blocks of TUI. Therefore, all attempts to model TUI were based on efforts to formalise the notion of TUI and the related terminology such as widgets or zones. A more complete list of terms can be found in the glossary 6.2.

In order to keep the modelling efforts comparable, a number of requirements were drafted in cooperation with Paul. These requirements spanned structural and behavioural aspects and detailed a few simple constraints. For example, a widget must have at least one handle or a handle may be bound. Such atomic requirement statements are of course far removed from what a customer might specify but the simple nature of the requirements was preferred as it meant that we could save time and focus on the essential, the modelling. A complete set of requirements can be found in the accompanying document, *"Scenario requirement description document"* which can be accessed at `http://tinyurl.com/cnqupjj`. An excerpt from the document containing all requirements can be found in Appendix B.

3.2.1 A simple TUI instance

The preliminary TUI model was the basis for choosing a set of widgets to be used to model the BPMN2 instance. The *"Capital Stock Management scenario description document"* document contained a list of all widgets that were thought to be useful during the study. From all these widgets, due to the reduction of the study scope to only model the first level of the scenario, only two widgets were finally used, Stamp and Chain.

Name	Physical Action	Intent	Result
Stamp [Create component]	The actor moves the widget onto the Toolbox, selects a component by activating the widget and then stamps one or multiple components onto the canvas.	The actor wants to select a component from the Toolbox and place it once or multiple times onto the canvas.	The desired number of components is placed on the canvas.
Stamp [Link components]	The actor moves the widget onto the Toolbox, selects a component by activating the widget and then stamps one or multiple components onto the canvas.	The actor wants to link existing components with the selected component or place it inside existing components by placing the focus point onto an existing component.	The existing components are linked by the new component or the new component is placed inside the new component.
Chain	The actor activates the widget on a component, selects a connector component by activating the widget and then slides the widget to a destination and activates it to confirm and select the kind of endpoint component.	The actor wants to create a new element by creating a link and endpoint from an existing component.	A new connector component and linked non-connector endpoint component are created.

Table 3.1: Table of widgets used in the study

Chapter 4

Case study

This chapter will guide the reader through study, from design of the measurements till the final conclusion. During the study, the tree previously identified visual modelling languages will be used in turns to try and model the scenario defined in 3. Before each modelling attempt, a few days were spent on getting acquainted with the languages, modelling some tutorials if available and consulting some papers highlighting examples.

The following sections will explain which measurements were collected in order to be able to evaluate the chosen visual modelling languages as well as explaining the point and weighting scheme of the measurements. Next will be a rundown of the actual study. While taking the study there were some problems which had to be addressed. These will be explained in the second to last section with the last section wrapping up the study and preparing for the next chapter which will concern itself with the evaluation.

4.1 Measurements

In order to be able to compare the different VMLs, measurements are needed that can be objectively collected and evaluated. Unfortunately the picking of these measurements is subjective and hence makes all attempts to be objective tainted. However, all subjective choices are given as is and no attempt was made to hide them. It is up to the reader to judge on the validity of the study.

The drafting of the measurements was accompanied by the *"Visual modelling Language Evaluation Criteria"* document which can be found at `http://tinyurl.com/bobus6c`. The measurements are split into six categories. *Tool support* is aimed to measure the availability and quality of language accompanying tool. The use of a tool is usually preferable as it usually offers many facilities that make modelling easier and less error prone. *Semantics & Transformation* captures the degree of formalism the VML offers. The *Expressivity* category captures the number of requirements that have been met by each language and computes a ratio. The *Usability* of each VML is measured by the category with the same name. *Error Checking* puts a number to the facilities each VML and its accompanying tools offer for rooting errors during the modelling process.

Finally *Validation & Verification* measures modularity, easing validation, and the facilities to proceed at a formal verification of the model. Below are the sub-categories for each of these main categories. Please consult *"Visual modelling Language Evaluation Criteria"* for more details.

Measurement categories	
Tool support	Availability Maintained Latest version Branch
Semantics & Transformation	Formally defined Transformability
Expressivity	# [X][1] # [X] satisfied # requirements partially satisfied # unsatisfied requirements Ratio
Usability	Naming conventions Naming fit Documentation Tutorial Hands-on-tutorial Primitive mutability Live suggestions
Error Checking	Time Syntax highlighting Degree Error correction suggestion Debugging possible
Validation & Verification	Modularity Verification scheme

Table 4.1: Table of measurements

These measurements were to be subjected to a weighting scheme. A small questionnaire was drafted to ask possible future users from the business domain how they would rank the six criteria. As of the writing of this document, not enough questionnaires have returned to objectively gauge the weight for each category. Therefore the sample has been broadened to include Software Engineers. More on the weighting can be read in section 5.7.

4.2 Study plan

As mentioned in 3.1.1, the scenario was split into three levels, each introducing more concepts of the BPMN2 language and using more metaphors and aspects of the TUI. In order to be able to compare the modelling performance

[1]Where X is either structural requirements, behavioural requirements or constraints.

of the languages given the scenario, a plan had to be followed. Therefore, it seemed unwise to simply take all requirements, as found in B, and model these sequentially. Therefore, the *"Capital Stock Management scenario description document"*'s tasks were taken as a basis to draft a scenario, level by level. For a definition of each requirement that will be stated by its abbreviation below, all requirements were included in Appendix B.

First, complete BPMN2 and TUI model were drafted in that order. To that end, all requirements pertaining to the BPMN2 B.1 and TUI B.3 model needed to be fulfilled. After these steps had been modelled, the process of modelling followed the level based separation of complexity. The modelling of the first level had eight steps, each stemming from identifying the task involved to fulfil the step.

4.2.1 Create Start Event

To create a Start Event we need a widget in Stamp mode. We therefore must fulfil **TMS1** and **TMS3**. As for widget behaviour, the Stamp requires **TMB1**, **TMB12**[2] and **TMB13**.

4.2.2 Create Activity

To state all requirements needed to create an activity we need to cater to both creation scenarios. The use of the Stamp is covered by the previous task so we can simply assume we need all those requirements too; **TMS1**, **TMS3**, **TMB1**, **TMB12** and **TMB13**. To use the Chain widget we also need **TMS2** and more items from**TMB12**. In addition we need to meet several requirements from the stock management model, depending on the type of activity that is created.

4.2.3 Check portfolio

In order to attribute the activity to a user said user and his portfolio are needed; **SS1**, **SS5** and **SS6**. Moreover, the related behaviour and constraints; **SB1** and **SC4** are needed.

4.2.4 Take action

Information about the customer, **SS1** and about an eventual order the customer may place, **SS4**, **SB2** and **SC1** is needed. To place an order, requirements catering to the subject of orders, stock, are needed too; **SS2** and **SS3**. Due to the encapsulation of information, details about the orders are not noted in BPMN2 and hence the differentiation between stocks for example would only become apparent if a decision is taken based on their difference.

4.2.5 Wait for response

Information about the customer, **SS1** and the action he may take; **SB4** is needed.

[2]For **TMB12**, only those rows are required that list the mode in question.

4.2.6 Review portfolio

In order to attribute the activity to a user said user and his portfolio are needed; **SS1**, **SS5** and **SS6**. Moreover, the related behaviour and constraints; **SB3** and **SC4** are needed.

4.2.7 Create Gateway / Create Event

Analogous to Create Activity.

4.2.8 Link components

To create a connection we need a widget in Stamp mode. We therefore must fulfil **TMS1** and **TMS3**. As for behaviour, the Stamp requires **TMB1**, **TMB12** and **TMB13**. To use the Chain widget we also need **TMS2** and more items from **TMB12**.

After having modelled the first level, most of the requirements were covered and it was deemed that the time investment needed to model the remaining two levels would not offer any significant new insights into the modelling languages.

4.3 Scope reduction

During the draft of the BPMN2 models, the scenario and all related items, the scenario was split into three levels which would symbolise the increase in complexity during a collaborative modelling approach. In theory, the customer would model his view and then the business people would model their end. Then the processes would get more and more detailed with all parties defining the back and forth that required for business processes. The small scenario that was drafted seemed to be not too complex to model quickly and gain some valuable insights.

Getting accustomed with several of the modelling languages proved more of a challenge than expected and time elapsed quickly. After having modelled the basis requirements and the first level for each language, the time allocated for the modelling process during the sturdy had elapsed. Seeing as the complexity of the modelling process would not increase from here on out, spending time on modelling the remaining levels would not benefit the study significantly.

Therefore it was chosen to reduce the scope of the study from initially three levels of complexity to two levels. The first level can be seen in Figure 3.1.1. The two remaining levels can be seen in Appendix C and Appendix D respectively.

4.4 Product

At the end of the modelling process, each language had produced multiple models which will be given in the subsequent sub-sections. The subjective experience gained during the modelling process will be reflected during the evaluation phase detailed in the next chapter. Each subsequent section dedicated to one particular language will include all models produced during the study. They will also include comments on the modelling experience and the models themselves.

4.4.1 UML + VOCL

VOCL serves as a visualisation for OCL hence the reason why structure and behaviour of all related domains are modelled in UML using the stand-alone version of UMLet[3] downloadable at `http://www.umlet.com/`. Figure4.2 shows the final result of this model after the entirety of the scenario's first level had been modelled. The constraints for this level are expressed in VOCL using an Eclipse VOCL-Plugin from the TU-Berlin. The plugin was developed during a student project and can be downloaded at `http://tfs.cs.tu-berlin.de/vocl/userguide/group1/`. The constraints can be seen in Figure 4.1.

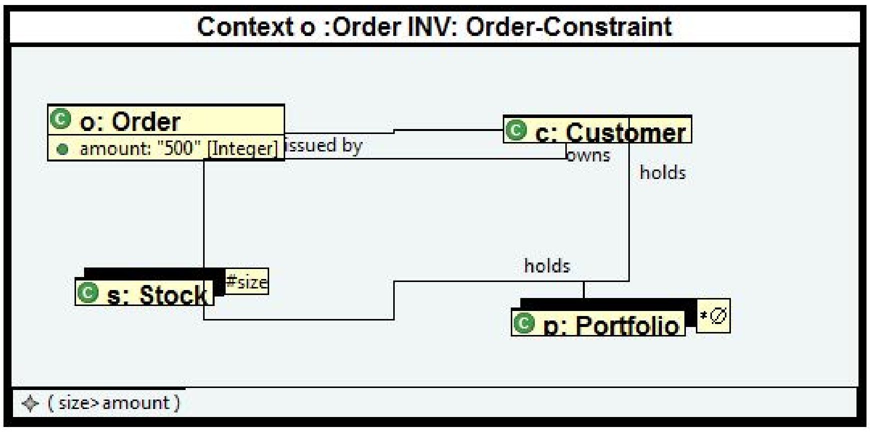


Figure 4.1: Constraints of the scenario's first level expressed in VOCL

The UML model uses different types of models to address structural and behavioural requirements. The models uses a vague packaging approach to separate concerns and then express their relation. The structure is expressed using Class Diagrams. User behaviour is modelled using an Use Case Diagram. The system behaviour is visualised using a Statechart. VOCL uses a visual representation close to that used in Class Diagrams to express the underlying OCL constraint.

[3]Version 11.4

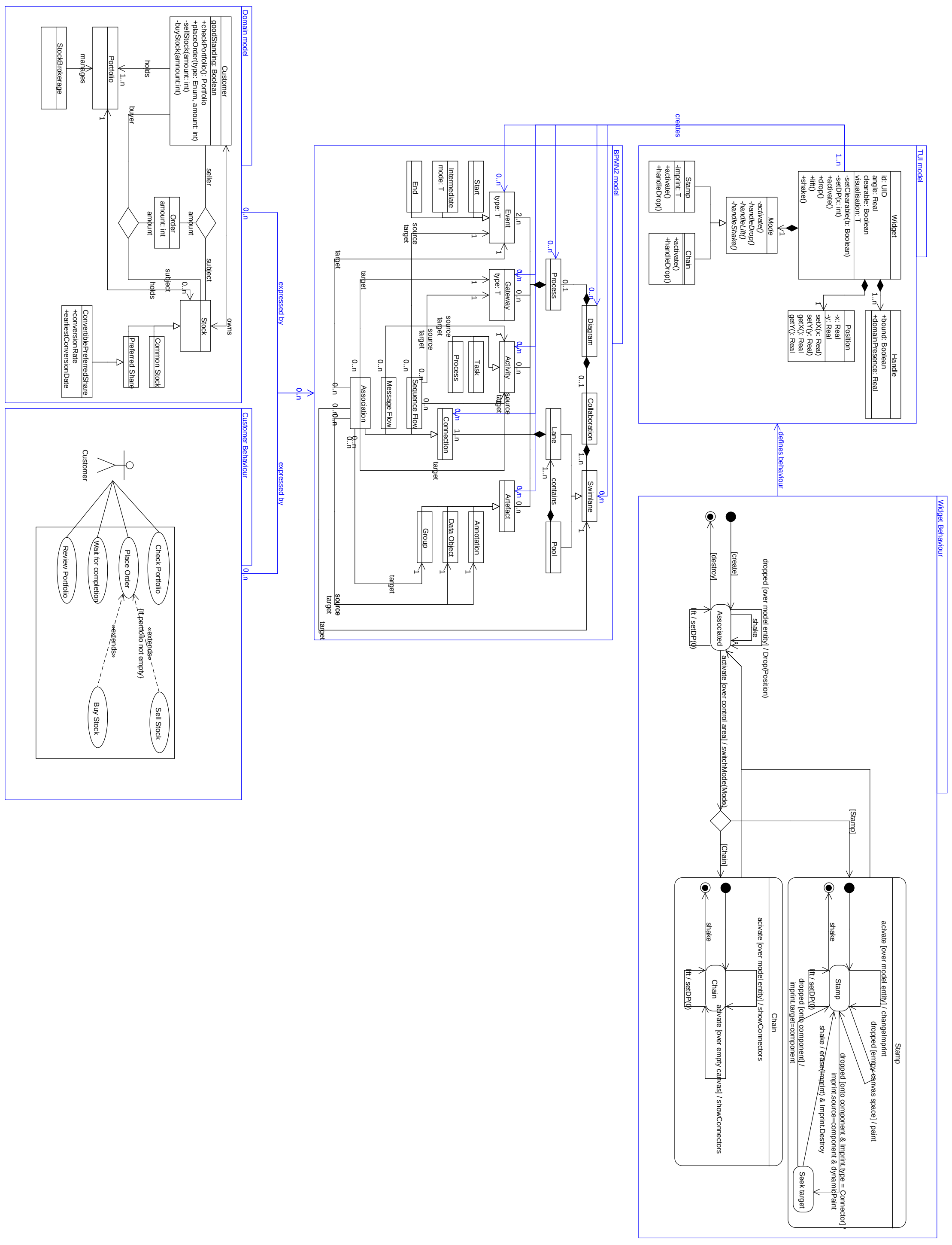


Figure 4.2: UML model for the first level of the scenario

4.4.2 VCL

The scenario modelled in VCL made use of the Visual Contract Builder (VCB), a tool for modelling VCL using an Eclipse plugin from the University of Luxembourg. The tool is available for download at `http://vcl.gforge.uni.lu/`. VCB uses a packaging mechanism which allows for the separation of concerns while modelling. VCL Structural Diagrams in the final stages for the TUI, BPMN2 and scenario can be found in Figure 4.3, 4.4, or 4.5 respectively.

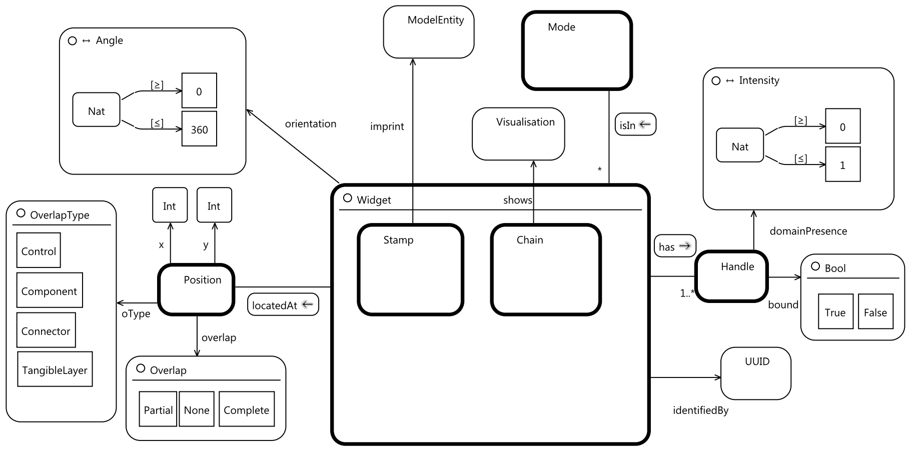


Figure 4.3: VCL Structure Diagram of the TUI.

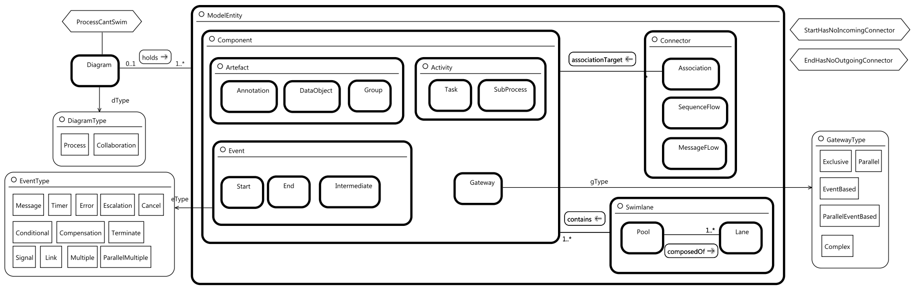


Figure 4.4: VCL Structure Diagram of the BPMN2 model.

The behaviour in VLC is modelled using Behavioural diagrams. Figure 4.6 and 4.7show these diagrams for the TUI and the scenario. The behaviour is expressed by refining the atomic behaviour such as the Stamp's Activate operation as shown in Figure 4.8. Update operations, those possibly producing state changes, are represented by contracts while query operations, those not modifying any states, are represented using assertions.

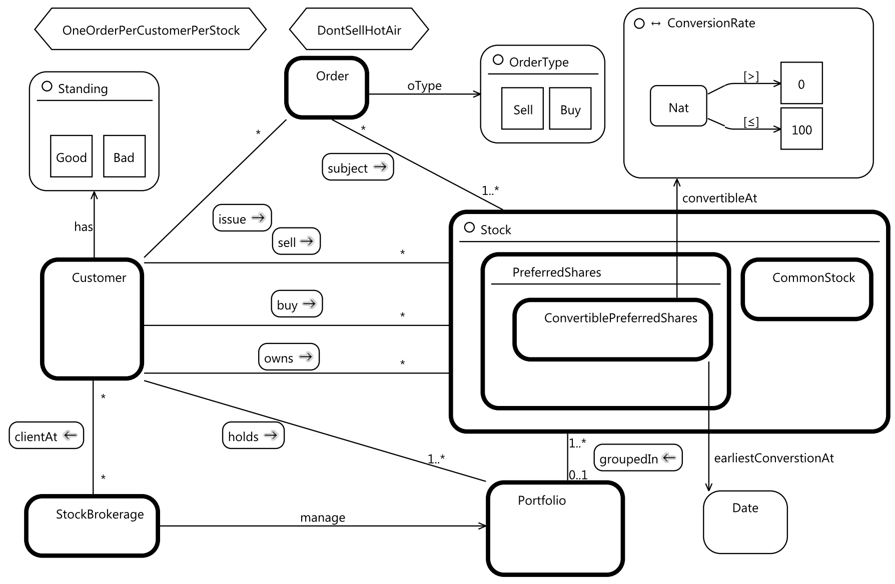


Figure 4.5: VCL Structure Diagram of the scenario.

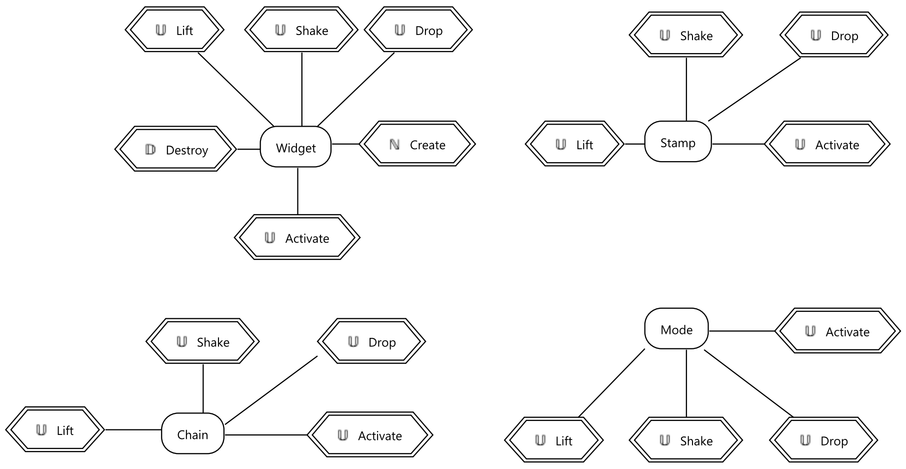


Figure 4.6: VCL Behaviour Diagram of the TUI model.

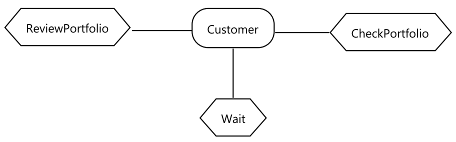


Figure 4.7: VCL Behaviour Diagram of the scenario.

Constraints, such as expressed by the invariants represented by the hexagonal boxes on the structural diagrams, are modelled using assertion diagrams.

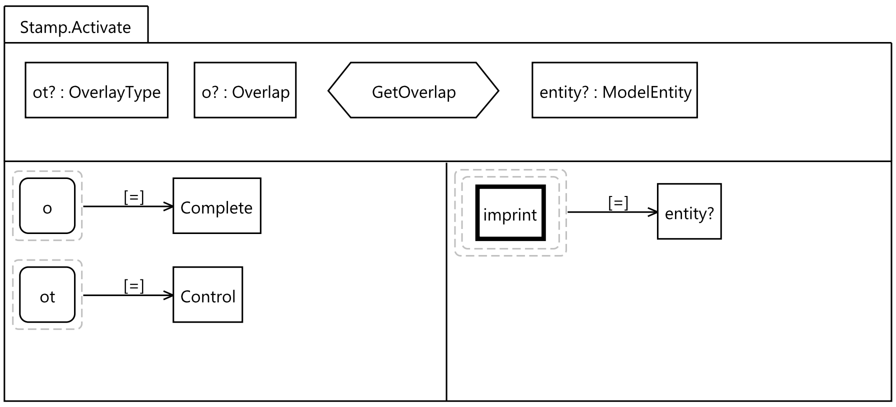


Figure 4.8: VCL Contract for the Stamp's local Activate operation.

Figure 4.9 for example shows the invariant that expressed the implicit constraint of BPMN2 models that Process Diagrams cannot hold Swimlanes which can only be used in Collaboration Diagrams.

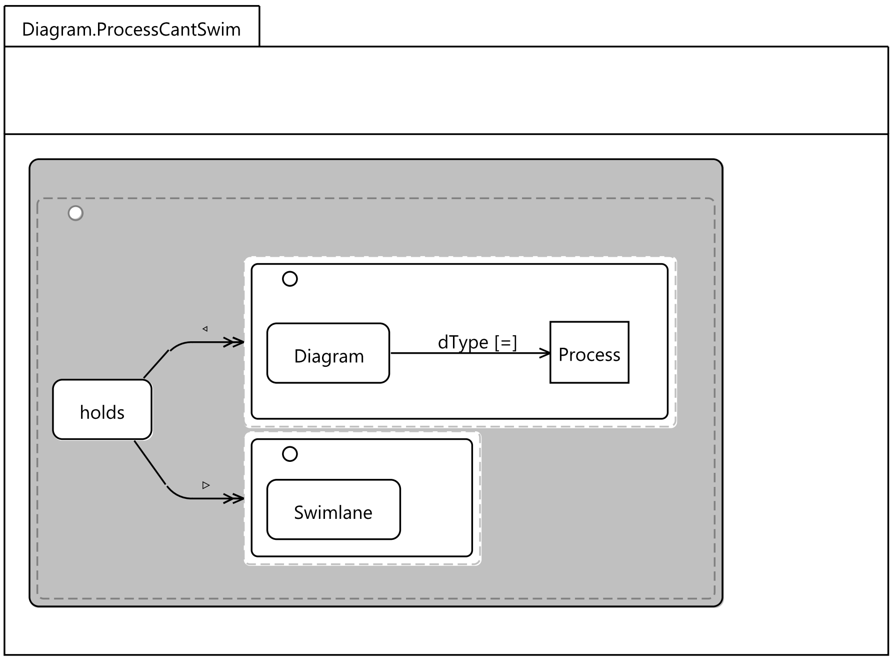


Figure 4.9: VCL Assertion of the ProcessCantSwim invariant.

4.4.3 Constraint Diagrams

Initially, a tool was used to attempt to model the scenario using Constraint Diagrams. The tool was the result of a joint project by the Universities of Kent and Brighton in the scope of a project to; *"[...]investigat[e] reasoning systems and tool for various visual constraint notations based on Venn and Euler diagrams"*. The tool as well as more information on the project can be found at `http://www.cs.kent.ac.uk/projects/rwd/`. The tool worked fine for small problem instances but with the model growing beyond even a handful of contours, the tool proved unusable due to an exponential increase in computing time to make even the simplest modifications. The problem persisting in all compatibility modes, it rendered the tool useless. Together with the fact that the tool had no support for the augmented language dialect, a more archaic form of modelling was adopted; drawing by hand. The revision of this document saw the hand drawing replaced by models created in Microsoft Visio 2010 with custom stencils.

The problem with drafting models by hand or with a simple drawing tool is that there is no mean to verify the model. This posed an obstacle that was rather hard to overcome as experts on the language are hard to come by. Another issue was the tidiness of the model. In order to avoid multiple lines crossing and a loss in readability, some concepts were simplified. Figure 4.10 shows the BPMN2 and TUI model using the augmented Constraint Diagrams notation. Figure 4.11 shows the modelling attempts to bring the TUI and BPMN2 models together to define how to use TUI instances to model BPMN2 instances. It also shows how behaviour is expressed using the augmented Constraint Diagrams language.

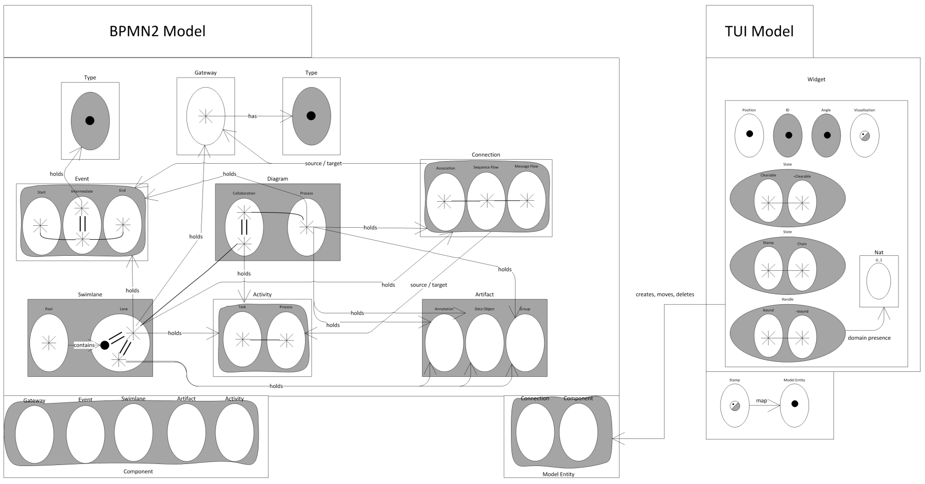


Figure 4.10: BPMN2 and TUI model using augmented Constraint Diagrams.

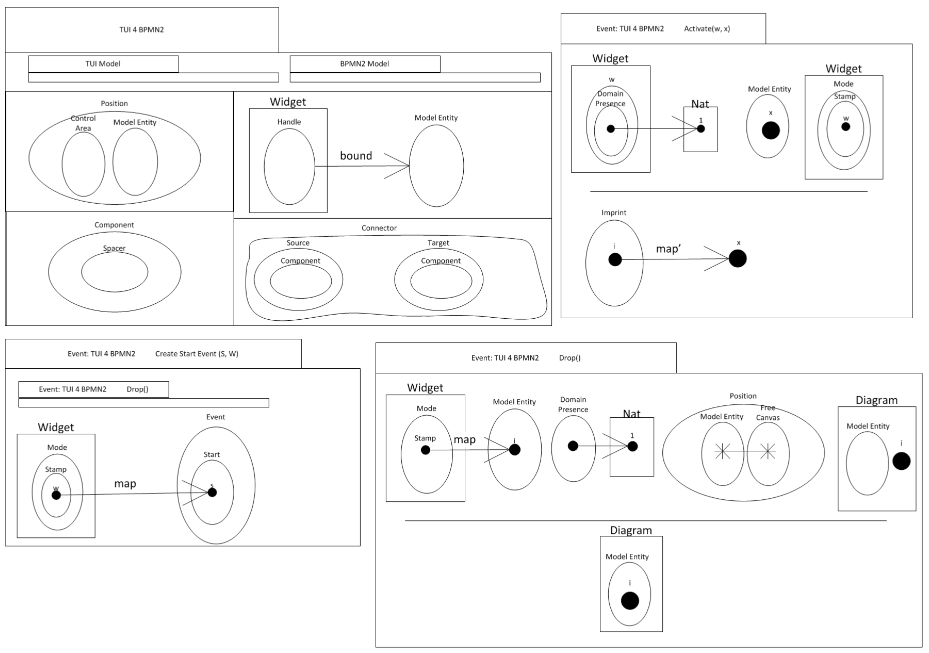


Figure 4.11: Behaviour and extension of concepts using augmented Constraint Diagrams.

Both diagrams use the notation defined by [42]. The rectangle labelled *Diagram* in the centre of Figure 4.10 for example denotes the type *Diagram* which can hold either a Collaboration or a Process. This is expressed by the sets placed inside the type. The disjointness is implicit by the sets not overlapping. The fact that a *Diagram* can only hold either a Collaboration or a Process is noted by the spider that has one foot in each set. The type being shaded means that the type is not defined for anything not belonging to one of the named sets therein.

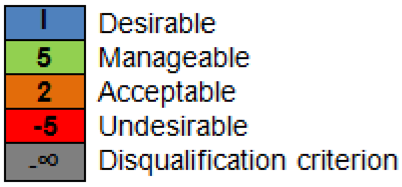
The behaviour, as shown in Figure 4.11 is given by the precondition, noted on top in the rectangle, and the postcondition, noted under the dividing line. What is unclear is how *"if-clause"* preconditions are formulated. The model assumes that only the behaviour whose precondition holds is executed and should multiple definitions exist for the same behaviour, that it is picked in a non-deterministic fashion.

Chapter 5

Evaluation

With the study completed, only its evaluation remained to be able to reach a final verdict. In this chapter, the measurement criteria as briefly described in section 4.1 are explained in depth. Each measure is rated on a scale that is associated with it. This scale is purely subjective and should not be interpreted otherwise. In the section after all measures have been introduced, a weighting scheme will be defined. It has been drafted by sending a questionnaire asking business experts and people working in the Software Engineering domain on a five point Likert scale how they would rate each criterion. The last section will comment some of the results and values.

The colour scale is associated with a point system according to the following table;



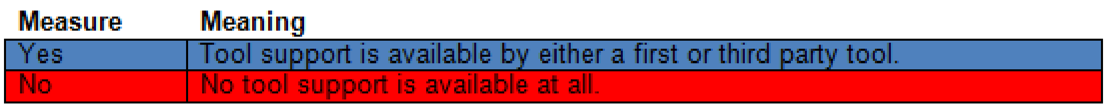
🟦	I	Desirable
🟩	5	Manageable
🟧	2	Acceptable
🟥	-5	Undesirable
⬜	$-\infty$	Disqualification criterion

5.1 Tool support

Automation of model transformation is only one of many advantages that tool support has. Without tool support there is a strong possibility that other key points like error checking are not possible or at least more cumbersome. Also, the lack of an adequate tool usually means an increase in development time and having to make compromises when it comes to representing the model in a visual format or an increase in effort required to deliver the same result. In order to evaluate the suitability of a tool, the following points are observed and evaluated.

5.1.1 Availability

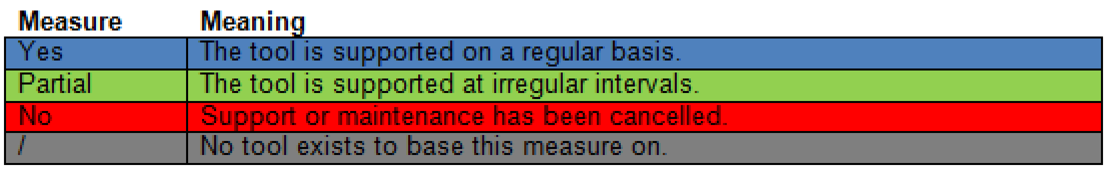
This criterion gauges whether tool support is available.



Measure	Meaning
Yes	Tool support is available by either a first or third party tool.
No	No tool support is available at all.

5.1.2 Maintainability

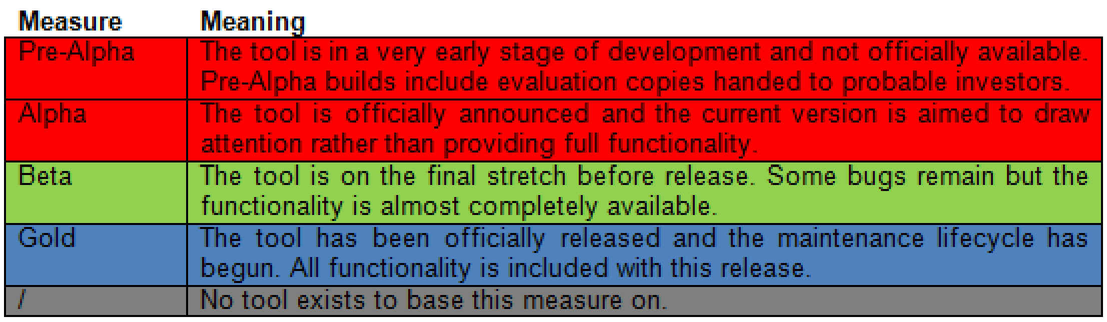
This criterion gauges whether an existing tool is supported by a team and maintained. A tool for which no information has been released for at least a year is considered to be no longer supported.



Measure	Meaning
Yes	The tool is supported on a regular basis.
Partial	The tool is supported at irregular intervals.
No	Support or maintenance has been cancelled.
/	No tool exists to base this measure on.

5.1.3 Latest version

This criterion simply collects data about the version of the tool.



Measure	Meaning
Pre-Alpha	The tool is in a very early stage of development and not officially available. Pre-Alpha builds include evaluation copies handed to probable investors.
Alpha	The tool is officially announced and the current version is aimed to draw attention rather than providing full functionality.
Beta	The tool is on the final stretch before release. Some bugs remain but the functionality is almost completely available.
Gold	The tool has been officially released and the maintenance lifecycle has begun. All functionality is included with this release.
/	No tool exists to base this measure on.

5.1.4 Branch

This criterion states branch that spawned the tool. This might be an interesting factor to judge the continuity plans for the tool and future versions, improvements and overall support.

Measure	Meaning
Academic	The tool was the result or by-product of academic research.
Research	The tool was the result or by-product of a research or standardisation effort by a research or special interest group.
Industrial - Open Source	The tool was the result of an industrial effort to generate revenue. The source code is available.
Industrial - Commercial	The tool was the result of an industrial effort to generate revenue. The source code is not available.
/	No tool exists to base this measure on.

5.2 Semantics & Transformation

In order to be able to transform a model into another model, say for generating a data model, the former would need to be expressed in a formal way to allow for a meaningful and coherent mapping or transformation. A lack of formalism could lead to problems with reproducibility and reuse of the language and transformation or mapping schemes.

5.2.1 Formally defined

This data point tells us if the model is formally defined or not.

Measure
Yes
Partially
No

5.2.2 Transformability

This data point measures whether models produced by the language are easily transformable as such and if the transformation is complete!

Measure	Meaning
Yes	Models produced by this language are easily transformable.
Partial	Model transformations can only be done partially or require a lot of effort.
No	Models produced by this language are not transformable or require effort that would go beyond simply redoing the model in another language better suited for transforming models.

5.3 Expressivity

The expressivity criteria try to capture data points which measure the performance of the VMLs to be able to compare their ability to correctly model the scenario. They measure these criteria by category; structural requirements, behavioural requirements, or constraints. Therefore, each of the following sections is measured thrice with the exception of the ratio which is not observed but computed.

5.3.1 # X (# X)

This data point measures the number of requirements for category X. Note that the absence of colour indicates that this measure takes a simple integer value.

Measure	Meaning
Integer	The number of requirements in this category.

5.3.2 # X satisfied (# Sat_X)

This data point measures the number of requirements of each category that have been satisfied.

Measure
🟦 >= 90%
🟩 75% <= x < 90%
🟧 50% <= x < 75%
⬛ < 50%

5.3.3 # requirements partially satisfied (# Part_Sat)

This data point measures the number of partially fulfilled requirements.

Measure	Meaning
Integer	The number of partially fulfilled requirements in this category.

5.3.4 # unsatisfied requirements (# UnSat)

This data point measures the number of unsatisfied requirements.

Measure	Meaning
Integer	The number of unfulfilled requirements in this category.

5.3.5 Ratio

This data point computes the ratio of satisfied requirements to the total number of requirements by Ratio $= \frac{\sum^{X} \#Sat\_X}{\sum^{X} \#X}$

Measure
1
0,9 <= x < 1
0,75 <= x < 0,9
0,5 <= x < 0,75
< 0,5

5.4 Usability

It is quite difficult to measure but important to know how easy it is to learn and use a new language. This is of course a highly subjective measure. Still, an objective approach can be tried using absolutes where possible in order to compare the languages in regard to usability. It is up to the reader to judge the chosen criteria.

5.4.1 Naming conventions

This measure states whether naming conventions are used and if to what degree. The adherence to naming conventions is important as it enables novel users to get used to the language and not confused by terminology.

Measure	Meaning
Core	Naming conventions are used for core language features only.
Complete	Naming conventions are used for core language features as well as any extensions the language offers.
Mixed	Naming conventions are sometimes followed and sometimes not.
None	No naming conventions are followed.

5.4.2 Naming fit

This measure captures whether names are well chosen and not misleading. Using easily comprehensible and meaningful names for concepts reinforces the effect of metaphors and abstractions.

Measure
Yes
Partially
No

5.4.3 Documentation

This measurement captures whether the language is documented and if then to what degree.

Measure	Meaning
Outline	The documentation only consists of a rough outline.
Core	The documentation details all core language features.
Complete	The documentation spans all language features.
None	These are not the droids you are looking for.

5.4.4 Tutorial

This data point captures if there is a tutorial and how extensive it is. Only disjoint examples which cover different scenarios are considered.

Measure
None
[1..3] examples
]3..5] examples
>5 examples

5.4.5 Hands-on tutorial

This data point captures if there is a hands-on tutorial and how extensive it is. Only disjoint examples which cover different scenarios are considered.

Measure
None
[1..3] examples
]3..5] examples
>5 examples

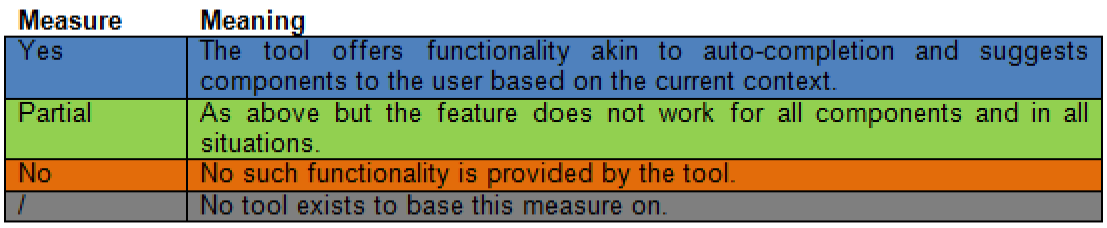
5.4.6 Primitive mutability

This measure notes if the use of language primitives depends on the context it is used in or if it is static.

Measure	Meaning
None	No mutability. All primitives are used in one context only and have only one meaning.
Low	Less than (inclusive) 10% of all primitives are mutable.
Medium	Less than 50% but more than 10% of all primitives are mutable.
High	More than (exclusive) 50% of all primitives are mutable.

5.4.7 Live suggestions

This criterion aims to capture the interactivity that the tool offers which might help users pilot the tool.



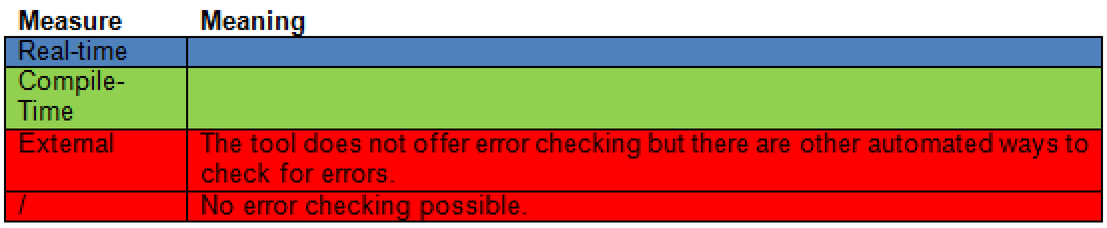
Measure	Meaning
Yes	The tool offers functionality akin to auto-completion and suggests components to the user based on the current context.
Partial	As above but the feature does not work for all components and in all situations.
No	No such functionality is provided by the tool.
/	No tool exists to base this measure on.

5.5 Error Checking

This criterion aims to evaluate the different levels of error checking the language might offer in conjunction with any tool there might be for the language.

5.5.1 Time

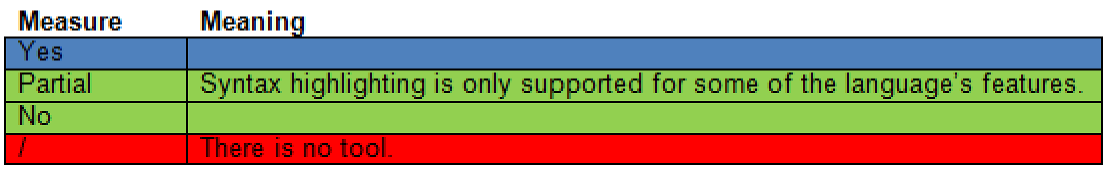
This data point notes at which kind of error checking is performed. Note that the first two options are only observable if the language is backed by a tool.



Measure	Meaning
Real-time	
Compile-Time	
External	The tool does not offer error checking but there are other automated ways to check for errors.
/	No error checking possible.

5.5.2 Syntax highlighting

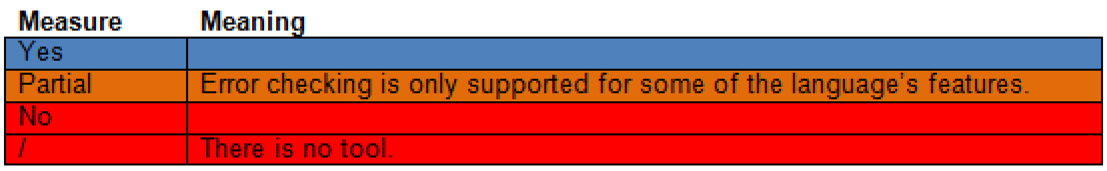
This measure captures if an existing tool uses syntax highlighting to provide feedback to the user about syntactical errors.



Measure	Meaning
Yes	
Partial	Syntax highlighting is only supported for some of the language's features.
No	
/	There is no tool.

5.5.3 Degree

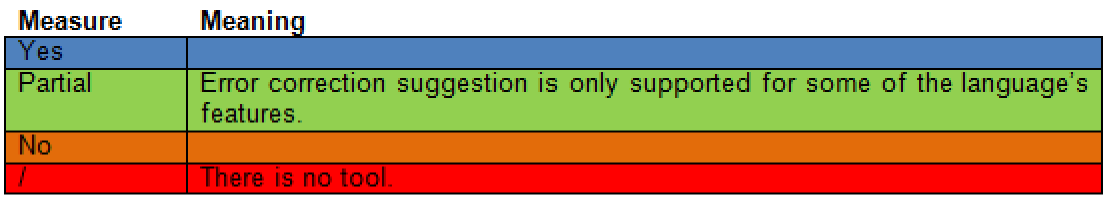
This data point states to what degree a tool offers error checking facilities.



Measure	Meaning
Yes	
Partial	Error checking is only supported for some of the language's features.
No	
/	There is no tool.

5.5.4 Error correction suggestion

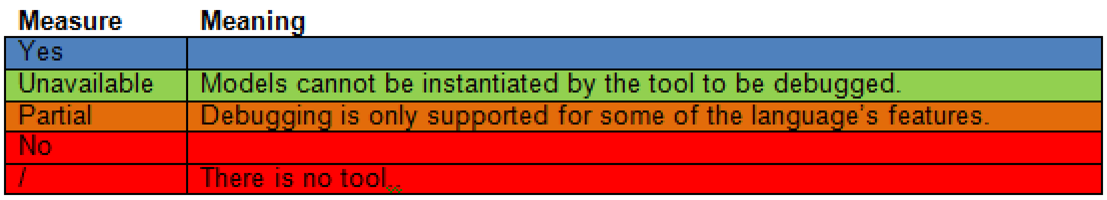
Tool support offers the means to suggest correct values to the user in case he should have made an error. This data point captures if such a functionality is offered by the tool.



Measure	Meaning
Yes	
Partial	Error correction suggestion is only supported for some of the language's features.
No	
/	There is no tool.

5.5.5 Debugging possible

This measure captures if an existing tool offers, and to what degree, to debug models.



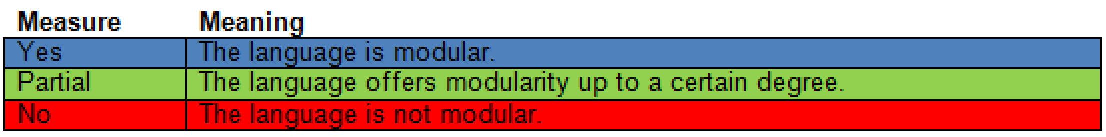
Measure	Meaning
Yes	
Unavailable	Models cannot be instantiated by the tool to be debugged.
Partial	Debugging is only supported for some of the language's features.
No	
/	There is no tool.

5.6 Validation & Verification

This criterion aims to evaluate the possibility to validate the model against the requirements after it has been completed. For most languages this step will have to be made manually by the clients. The reason is simple, if the requirements were formal enough to be able to be checked against a model than they could be used to produce said model in the first place. Nevertheless, the language can offer some tools to statically or dynamically verify the soundness of the model. The basic option would be the verification by a static tool akin to the verification scheme of an XML document. The next best step would be the real time error checking that should guarantee the soundness of the final model. More advanced techniques then allow for instantiation of the model which makes it easier to visually see quirks and kinks in the model that are not in line with the requirements. The pinnacle of functionality that can be provided is an interactive model explorer that does not only generate random model instances but allows for the user to explore and build instances as he sees fit.

5.6.1 Modularity

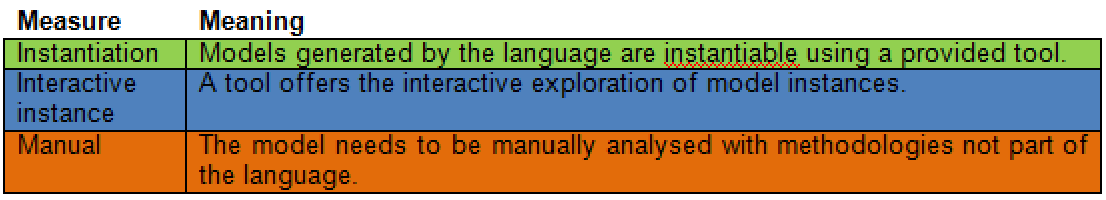
This measure aims to capture the modularity of the language. Is there a clear way to implement separation of concerns through use of for example packages or simply using different files which need to be imported? Modularity is a desirable quality as it gives a clear option to separate concerns.



Measure	Meaning
Yes	The language is modular.
Partial	The language offers modularity up to a certain degree.
No	The language is not modular.

5.6.2 Verification scheme

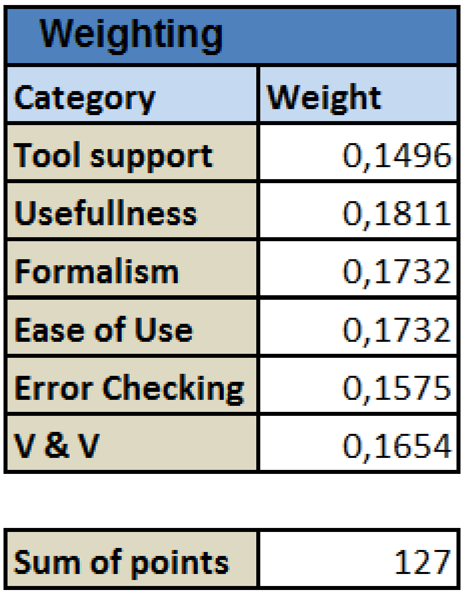
This measure states which scheme the language or related tools offer to verify the model.



Measure	Meaning
Instantiation	Models generated by the language are instantiable using a provided tool.
Interactive instance	A tool offers the interactive exploration of model instances.
Manual	The model needs to be manually analysed with methodologies not part of the language.

5.7 Weighting scheme

In an attempt to make the final outcome of the study less dependent on the author and introduce more objective measurement weighting, a questionnaire was devised. A sample of the questionnaire can be found in Appendix E. The returned questionnaires were evaluated by attributing one vote per item per intensity on the Likert scale. *"Not important at all"* was worth one point whereas *"Very important"* was worth five points. The sum of votes for each category were then divided by the total number of votes. The result can be seen in Table 5.7.



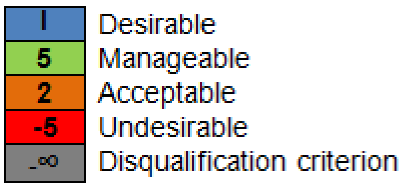
Weighting	
Category	**Weight**
Tool support	0,1496
Usefullness	0,1811
Formalism	0,1732
Ease of Use	0,1732
Error Checking	0,1575
V & V	0,1654

Sum of points	127

The distribution of returned questionnaires was heavily skewed in favour of Software Engineers with only one fifth of all questionnaires stemming from business experts. Unfortunately, the number of questionnaires that were returned was also quite poor with only 6 questionnaires being sent back in total.

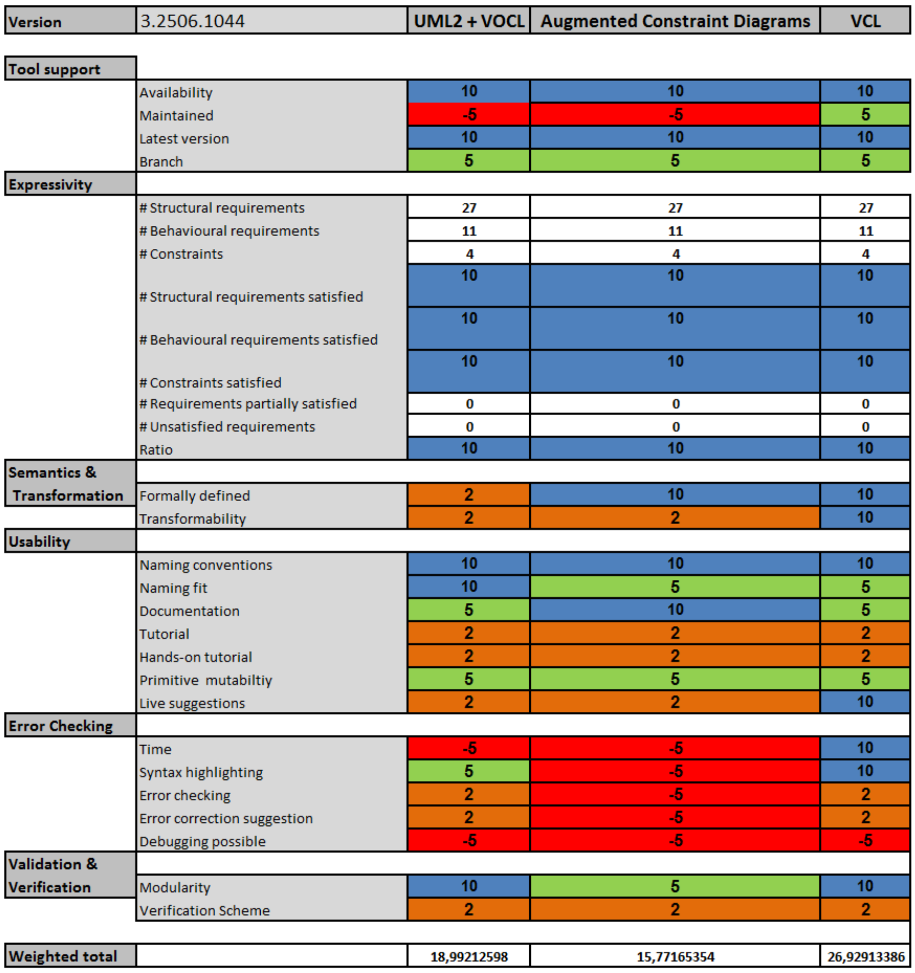
5.8 Results

All results are compiled by the coloured scale as defined by Figure 5, reprinted for the readers convenience below.



🟦 I	Desirable
🟩 5	Manageable
🟧 2	Acceptable
🟥 -5	Undesirable
⬛ $-\infty$	Disqualification criterion

In order to facilitate the computation of the final results, an Excel table was used. Table 5.8 shows the findings of the study. For a list of requirements that were considered for the expressivity criterion, consult Appendix B. As can be seen in the table, the victor by points is the Visual Constraint Language. The following paragraphs will comment on the results and highlight interesting findings.



Version	3.2506.1044	UML2 + VOCL	Augmented Constraint Diagrams	VCL
Tool support				
	Availability	10	10	10
	Maintained	-5	-5	5
	Latest version	10	10	10
	Branch	5	5	5
Expressivity				
	# Structural requirements	27	27	27
	# Behavioural requirements	11	11	11
	# Constraints	4	4	4
	# Structural requirements satisfied	10	10	10
	# Behavioural requirements satisfied	10	10	10
	# Constraints satisfied	10	10	10
	# Requirements partially satisfied	0	0	0
	# Unsatisfied requirements	0	0	0
	Ratio	10	10	10
Semantics & Transformation				
	Formally defined	2	10	10
	Transformability	2	2	10
Usability				
	Naming conventions	10	10	10
	Naming fit	10	5	5
	Documentation	5	10	5
	Tutorial	2	2	2
	Hands-on tutorial	2	2	2
	Primitive mutabiltiy	5	5	5
	Live suggestions	2	2	10
Error Checking				
	Time	-5	-5	10
	Syntax highlighting	5	-5	10
	Error checking	2	-5	2
	Error correction suggestion	2	-5	2
	Debugging possible	-5	-5	-5
Validation & Verification				
	Modularity	10	5	10
	Verification Scheme	2	2	2
Weighted total		18,99212598	15,77165354	26,92913386

Taking a close look at the *"Tool support"* criteria, one major difference between all tools can be identified, the degree of maintenance. While all languages have at least one tool, only the one offered for VCL is still maintained. All other tools were the results of projects that have since concluded and are hence no longer maintained. With VCB, the tool for VCL, also hailing from the academic sector, it might share the same fate if the language fails to reach the critical mass to persist beyond the current project. Please note that for the language duo UML+VOCL, only tools for VOCL were evaluated as UML offers a myriad tools in various forms and more than enough are adequate for the task. This is a major benefit if one uses an accepted standard.

The *"Expressivity"* criteria offer little room for interpretation. All requirements that had been dressed have also been satisfied, hence the good ratio. The author would like to not that it was not possible, lacking experts in the field, to verify the models written using VOCL and Constraint Diagrams. The tools offering no support in that matter also did not help in avoiding mistakes. The author acknowledges that he has possibly made multiple mistakes but that he would like to give any language the benefit of the doubt which means allocating full points. With the VCB tool offering live error checking, less mistakes have probably be made in VCL although a final verification by an expert would still be advisable.

"Expressivity" was straight forward to evaluate. With UML under criticism for not being thoroughly formal [44] it also did not reach the full score for transformability. While some diagram types can without much effort be transformed, this cannot be said of all of them. While the core Constraint Diagrams can be translated into OCL and then further, the augmented version lacks that possibility.

While UML is very well documented and offers a huge amount of all kinds of tutorials, VOCL is lacking in those disciplines. While only very few tutorials exist, the documentation is usable but not complete. The naming conventions and fit on the other hand leave nothing to be desired. The augmented version of the Constraint Diagrams are well documented and offer a couple of tutorials. However, the language uses some rather mathematical or domain specific terminology which might complicate matters for non-experts in the field. The same holds true for VCL. While the language is well documented, the VCB could offer more stand-alone documentation even though the build-in suggestion system makes up it to some degree.

"Error Checking" criteria are highly dependent on the use of an automated process or tool. With the tool for the augmented Constraint Diagrams failing to work properly and the models being drafted by hand, there was no way to have any error checking facilities. Only very few UML tools offer error checking and so does the VOCL tool. The conversion to OCL allows for the use of some error finding methodology and the tool itself restricts the syntactical constructs that can be visually expressed thereby somehow forcing the user to not make mistakes. VCB offers real rime error checking and details about the nature of the error for most of the diagram types. None of the languages or their tools can instantiate models which would have helped in the validation and verification process.

VOCL benefits from UMLs packaging facilities to separate concerns and express models in a modular fashion. VCL also applies a packaging scheme. While it is possible to extend diagrams in the augmented Constraint Diagrams, thereby separating concerns, it seems more cumbersome than the packaging mechanisms offered by the other two languages. Without any possibility to instantiate the models, all models produced by the languages will have to be verified and validated manually.

Chapter 6

Conclusion

After an initial broad selection of all related material, only a good two dozen resources were deemed relevant. The dozen resources that remained after further selection could be grouped in three camps, one routing for the Visual Object Constraint Language (VOCL), one for Constraint Diagrams and one for the Visual Constraint Language (VCL). VOCL required the use of UML, Constraint Diagrams, not expressive enough on their own, were used in an extended version. VCL was used as is. After selecting adequate tools for the modelling process, the different advantages and disadvantages of the languages and their tools became clear.

With all tools except for the Visual Constraint Builder, VCL's tool, being either unusable or not up to speed with that modern tools offer in regard of usability and support, VCL was the clear victor in that regard. UML in conjunction with VOCL proved adequate to model the BPMN2 Stock Management scenario. The use of VOCL however was rather cumbersome and error prone. Modelling using the augmented Constraint Diagrams easily and quickly advanced which was due to the fact that it was done by hand. Unfortunately this also meant no support or error checking. With the language not being broadly established and having no expert at hand it has yet to be determined if the produced model is correct. With VCL, the modelling process proved to be very modular with the language separating structure, behaviour and constraints clearly even from a modelling point of view.

Having gathered all measurements and weighted the scores for each evaluation criteria, the augmented Constraint Diagrams took third place mainly due to the lack of tool support and hence error checking capabilities. The effects on productivity were not measured although the drafting of models by hand on paper will have its problems in a collaborative industrial or business environment. Second place was claimed by the tandem of UML and VOCL. Wile UML performed admirably, there is a reason after all why it is so successful, VOCL proved to be no driving force to propel the team further. The winner is VCL, offering a well defined language that could offer a better naming scheme but offers not much else to criticise. The VCB tool did help enormously in cementing VCL's gold medal but without continued support for the tool the edge that VCL currently holds over the other languages might quickly become dull.

6.1 Future work

The next steps of the thesis involve a refinement of the scenario in, hopefully, close collaboration with business actors. Then the modelling process in VCL will produce models of the BPMN2 specification and the TUI framework. After these have been completed they will be intertwined such that a model of the modelling process of the BPMN2 scenario using TUI can be generated.

6.2 Long time goals

In the long run it is thought to use a visual modelling language to express all models needed for further modelling processes using TUI. For example, a model of the human biology could be drafted on the table using VCL and then used to instantiate different models illustrating concepts directly on the table. That way, the modelling process could completely take place in the collaborative TUI environment without the need to first plug in external models which would not be guaranteed to be compatible as is. Also, extensions and modifications would be easy to realise. These goals are a vision of what could be, well outside the scope of the author's Master's Thesis.

Bibliography

[1] Eva Hornecker and Jacob Buur. Getting a grip on tangible interaction: a framework on physical space and social interaction. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, CHI '06, pages 437–446, New York, NY, USA, 2006. ACM.

[2] D. Spinellis. UML Everywhere. *Software, IEEE*, 27(5):90 –91, sept.-oct. 2010.

[3] Object Management Group. Object Management Group Business Process Model and Notation, Version 2.0. Technical report, Object Management Group.

[4] John Hunt. Agile methods and the agile manifesto. In *Agile Software Construction*, pages 9–30. Springer London, 2006. 10.1007/1-84628-262-4_2.

[5] F Boers. Metaphor awareness and vocabulary retention. *Applied linguistics*, 21(4):553–571, December 2000.

[6] Sharon Alayne Widmayer. Schema theory: An introduction. *Retrieved April*, 2005.

[7] Google. Google Scholar, 2012.

[8] Consortium Luxembourg. Findit.lu, 2012.

[9] Google. Google Luxembourg, 2012.

[10] XJ Technologies Company. Simulation Software Tool - AnyLogic, 2012.

[11] Inc. Ventana Systems. Vensim, 2012.

[12] Nicolas Genon, Daniel Amyot, and P. Heymans. Analysing the Cognitive Effectiveness of the UCM Visual Notation. *System Analysis and Modeling: About Models*, 6598/2011:221–240, 2011.

[13] D Varro. Towards Symbolic Analysis of Visual Modeling Languages. *Electronic Notes in Theoretical Computer Science*, 72(3):51–64, February 2003.

[14] Andrew Fish, Jean Flower, and John Howse. The semantics of augmented constraint diagrams. *Journal of Visual Languages & Computing*, 16(6):541–573, December 2005.

[15] Object Management Group. Object Management Group - UML.

[16] Joseph Yossi Gil, John Howse, and Stuart Kent. Constraint Diagrams : A Step Beyond UML. In *Technology of Object-Oriented Languages and Systems (TOOLS USA'99)*, Santa Barbara, California , USA, 1999.

[17] Robert Muetzelfeldt and Jon Massheder. The Simile visual modelling environment. *European Journal of Agronomy*, 18(3-4):345–358, January 2003.

[18] S. Sadi and P. Maes. subTextile: Reduced event-oriented programming system for sensate actuated materials. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007)*, pages 171–174, September 2007.

[19] James Lin, Michael Thomsen, and J.A. Landay. A visual language for sketching large and complex interactive designs. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Changing our world, changing ourselves*, number 4, pages 307–314, New York, New York, USA, 2002. ACM.

[20] A. Schurr, A Winter, and A. Zundorf. Visual programming with graph rewriting systems. In *Visual Languages, Proceedings., 11th IEEE International Symposium on*, pages 326–333, Darmstadt , Germany, 1995. IEEE.

[21] L. Spratt and A. Ambler. A visual logic programming language based on sets and partitioning constraints. In *993 IEEE Symposium on Visual Languages*, pages 204–208. IEEE Comput. Soc. Press, 1993.

[22] Margaret Burnett, John Atwood, R. Walpole Djang, J. Reichwein, H. Gottfried, and S. Yang. Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm. *Journal of functional programming*, 11(2):155–206, 2001.

[23] D. Lucanin and Ivan Fabek. A visual programming language for drawing and executing flowcharts. In *MIPRO, 2011 Proceedings of the 34th International Convention*, pages 1679–1684. IEEE, 2011.

[24] Jonathan Sprinkle and Gabor Karsai.

[25] KD Swenson. A visual language to describe collaborative work. In *Visual Languages, 1993., Proceedings 1993 IEEE Symposium on*, pages 298–303, Bergen, 1993. IEEE.

[26] Roswitha Bardohl, Hartmut Ehrig, J. De Lara, and G. Taentzer. Integrating meta-modelling aspects with graph transformation for efficient visual language definition and model manipulation. *Fundamental Approaches to Software Engineering*, pages 214–228, 2004.

[27] I. Burnett, M.J. Baker, C. Bohus, P. Carlson, S. Yang, and P. Van Zee. Scaling Up Visual Programming Languages. *Computer*, 28(3):45–54, March 1995.

[28] Akos Schmidt and D. Varró. CheckVML: A tool for model checking visual modeling languages. *UML 2003 - The Unified Modeling Language. Modeling Languages and Applications*, 2863:92–95, 2003.

[29] Sara Brockmans, Raphael Volz, and Andreas Eberhart. Visual modeling of OWL DL ontologies using UML. In *The Semantic WebISWC 2004*, pages 198–213. Springer, 2004.

[30] Terry Halpin. Object-role modeling: an overview. In *In http://www.orm.net/pdf/ORMwhitePaper.pdf*, 1998.

[31] Emmanuel Chailloux and Philippe Codognet. Toward visual constraint programming. *Visual Languages, 1997. Proceedings. 1997 IEEE Symposium on*, pages 420–421, 1997.

[32] Object Management Group. Object Constraint Language, 2006.

[33] Visual OCL.

[34] C. Kiesner, G. Taentzer, and J. Winkelmann. Visual OCL: A Visual Notation of the Object Constraint Language. Technical Report 2002/23, Technical University of Berlin, 2002.

[35] Paolo Bottoni, Manuel Koch, Francesco Parisi-presicce, and Gabriele Taentzer.

[36] M. Lohmann, S. Sauer, and G. Engels. Executable Visual Contracts. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, pages 63–70. IEEE, 2005.

[37] Nuno Amálio and Pierre Kelsen. VCL, a visual language for modelling software systems formally. *Computer*, pages 282–284, August 2010.

[38] N. Amalio and Pierre Kelsen. Modular design by contract visually and formally using VCL. In *Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium on*, pages 227–234. IEEE, September 2010.

[39] Nuno Amáio, Pierre Kelsen, Qin Ma, and Christian Glodt. Using VCL as an aspect-oriented approach to requirements modelling. *Transactions on Aspect-Oriented Software Development*, 7:151–199, 2010.

[40] Nuno Amálio, Christian Glodt, and Pierre Kelsen. Building VCL Models and Automatically Generating Z Specifications from Them. *FM 2011: Formal Methods*, 6664:149–153, 2011.

[41] Stuart Kent. Constraint diagrams: visualizing invariants in object-oriented models. In *OOPSLA '97 Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, volume 32. ACM, 1997.

[42] John Howse, Steve Schuman, Gem Stapleton, and Ian Oliver. Diagrammatic Formal Specification of a Configuration Control Platform. *Electronic Notes in Theoretical Computer Science*, 259:87–104, December 2009.

[43] G. Stapleton. A Decidable Constraint Diagram Reasoning System. *Journal of Logic and Computation*, 15(6):975–1008, December 2005.

[44] D. Milicev. On the Semantics of Associations and Association Ends in UML. *Software Engineering, IEEE Transactions on*, 33(4):238 –251, april 2007.

Glossary

Application Context, App Context This is the context of the application giving a specific semantics to each tool or function used. It is makes use of the underlying widget actions of level 2.

Binding, Strong binding Instantiation (M0) of specific Widget (M1) on a physical handle. We can now use the widget in the application context. This is done by the binding effect. Not to be confused with selection.

Canvas View Point, View port It is the view port (zoom, orientation. size) on the infinite canvas as displayed by the application. Comparable to zones on the Canvas layer.

Canvas, Canvas layer Application Context layer on which the application draws on.

Component A non-connector model entity. Usually a box or circle shape representing a concept rather than a relation.

Connector A non-component model entity usually in the shape of an arrow tipped line that links components and creates logical relations.

Fixed Control Points, FCP Virtual handles appearing next to bindable entities on the surface. E.g. Circle which can be linked to a puck by placing it inside. They are located on the Tangible layer and are implicitly there (e.g. displayed upon proximity detection).

Focus area Same as Focus line but as a real area.

Focus horizon Same as Focus point but as a line.

Focus point Single point or small area (area only big enough to accommodate a widget, still treated as a point) on which a single widget focuses (e.g. during zoom action)

Handle, Physical entity Physical counterpart of the VOb, often the physical part(s) of a widget.

Interaction A specific way of influencing a control limited by some constraints due to its specific purpose. See TAC for constraint description. E.g. moving a puck around or activating a button.

Level 1 Actions Simple atomic actions.

Level 2 Actions Real usable actions by the user having a tangible semantics.

Manipulation Superset of Interaction but the way of influencing the token or container (data) dictates the purpose. E.g. Flipping a box upside down to empty the linked virtual container.

Model entity They are VObs that are used as primitives in the application context. All entities available from the application toolbox.

Reality Domain encompassing both the virtual/intangible and the physical/tangible domain.

Selection, weak binding, multiple binding For an already bound widget multiple selections are possible, those selections are weakly linked to the widget itself. Done by the selection effect.

Surface The conceivable work surface of the table. Root zone, divided into different layers

Tangible layer The layer on which widget virtual structures are drawn on as well as specific zones. Visually independent of the underlying application Context layer though zones usually represent tangible areas for application interaction.

The actor Person standing at the surface, actively manipulating widgets to achieve a goal.

Tool box A zone whose sub-areas represent different tools which can be bound and used within the application context.

Virtual object, VOb Any virtual object representing a virtual entity, e.g. control, data, tool. Each VOb can be bound by a widget via a zone on the tangible layer.

Widget Behaviour Information describing the manipulation possibilities as well as the pre- and post-state of an action.

Widget Semantics Part of the semantics is expressed via effect. But the application context is also providing part of the meaning as an application specific function/intention.

Widget Structure Information about the appearance and what visual, haptic, audible input and feedback is required to achieve full functionality.

Widget Syntax The set of syntactically valid widget models. Both correctly assembled and mapped to actions.

Zones Planar sub-divisions of the tangible layer representing special zones of interest such as Fixed Control Points. Zones enable the widget to transparently switch between expected behaviour (program functions) dictated by the App context.

Appendix A

List of literature resources

- Amlio, N., & Kelsen, P. (2010). Modular design by contract visually and formally using VCL. Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium on (pp. 227234). IEEE. doi:10.1109/VLHCC.2010.39

- Amlio, N., Glodt, C., & Kelsen, P. (2011). Building VCL Models and Automatically Generating Z Specifications from Them. (M. Butler & W. Schulte, Eds.)FM 2011: Formal Methods, 6664, 149-153. Springer Berlin Heidelberg. doi:10.1007/978-3-642-21437-0

- Amio, N., Kelsen, P., Ma, Q., & Glodt, C. (2010). Using VCL as an aspectoriented approach to requirements modelling. Transactions on Aspect Oriented Software Development, 7, 151-199. doi:10.1007/978-3-642-16086-8_5

- Bardohl, R., Ehrig, H., De Lara, J., & Taentzer, G. (2004). Integrating meta-modelling aspects with graph transformation for efficient visual language definition and model manipulation. Fundamental Approaches to Software Engineering, 214228. Springer.

- Bottoni, P., Koch, M., Parisi-presicce, F., & Taentzer, G. (2001). A Visualization of OCL using Collaborations. (M. Gogolla & C. Kobryn, Eds.)UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools, 2185, 257-271. Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/3-540-45441-1

- Brockmans, S., Volz, R., & Eberhart, A. (2004). Visual modeling of OWL DL ontologies using UML. The Semantic WebISWC 2004 (pp. 198213). Springer.

- Burnett, I., Baker, M. J., Bohus, C., Carlson, P., Yang, S., & Van Zee, P. (1995). Scaling Up Visual Programming Languages. Computer, 28(3), 45-54. doi:10.1109/2.366157

- Burnett, M., Atwood, J., Walpole Djang, R., Reichwein, J., Gottfried, H., & Yang, S. (2001). Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm. Journal of functional programming, 11(2), 155206. Cambridge University Press.

- Chailloux, E., & Codognet, P. (1997). Toward visual constraint programming. Visual Languages, 1997. Proceedings. 1997 IEEE Symposium on, 420-421.
- Genon, N., Amyot, D., & Heymans, P. (2011). Analysing the Cognitive Effectiveness of the UCM Visual Notation. System Analysis and Modeling: About Models, 6598/2011, 221240. Springer. Retrieved from http://www.springerlink.com/index/G7746GM683W97620.pdf
- Gil, J. Y., Howse, J., & Kent, S. (1999). Constraint Diagrams : A Step Beyond UML. Technology of Object-Oriented Languages and Systems (TOOLS USA99). Santa Barbara, California , USA. Retrieved from http://kar.kent.ac.uk/id/eprint/21740
- Halpin, T. (1998). Object-Role Modeling: an overview.
- Howse, J., Schuman, S., Stapleton, G., & Oliver, I. (2009). Diagrammatic Formal Specification of a Configuration Control Platform. Electronic Notes in Theoretical Computer Science, 259, 87-104. doi:10.1016-/j.entcs.2009.12.019
- Kent, S. (1997). Constraint diagrams: visualizing invariants in object-oriented models. OOPSLA 97 Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (Vol. 32). ACM.
- Lin, J., Thomsen, M., & Landay, J. A. (2002). A visual language for sketching large and complex interactive designs. Proceedings of the SIGCHI conference on Human factors in computing systems: Changing our world, changing ourselves (pp. 307314). New York, New York, USA: ACM. doi:10.1145/503429.503431
- Lohmann, M., Sauer, S., & Engels, G. (2005). Executable Visual Contracts. 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC05) (pp. 63-70). IEEE. doi:10.1109/VLH-CC.2005.35
- Lucanin, D., & Fabek, I. (2011). A visual programming language for drawing and executing flowcharts. MIPRO, 2011 Proceedings of the 34th International Convention (pp. 16791684). IEEE.
- Muetzelfeldt, R., & Massheder, J. (2003). The Simile visual modelling environment. European Journal of Agronomy, 18(3-4), 345-358. doi:10.1016/S1161-0301(02)00112-0
- Group, O. M. (2006). Object Constraint Language - OMG Available Specification - Version 2. OMG.
- Sadi, S., & Maes, P. (2007). subTextile: Reduced event-oriented programming system for sensate actuated materials. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007), 171-174. Ieee. doi:10.1109/VLHCC.2007.37
- Schmidt, A., & Varr, D. (2003). CheckVML: A tool for model checking visual modeling languages. UML 2003 - The Unified Modeling Language. Modeling Languages and Applications, 2863, 9295. Springer.

- Schurr, A., Winter, A., & Zundorf, A. (1995). Visual programming with graph rewriting systems. Visual Languages, Proceedings., 11th IEEE International Symposium on (pp. 326333). Darmstadt , Germany: IEEE.
- Spratt, L., & Ambler, A. (1993). A visual logic programming language based on sets and partitioning constraints. 993 IEEE Symposium on Visual Languages (pp. 204-208). IEEE Comput. Soc. Press. doi:10.1109/VL.1993.269597
- Sprinkle, J., & Karsai, G. (2004). A domain-specific visual language for domain model evolution. Journal of Visual Languages & Computing, 15(3-4), 291-307. doi:10.1016/j.jvlc.2004.01.006
- Stapleton, G. (2005). A Decidable Constraint Diagram Reasoning System. Journal of Logic and Computation, 15(6), 975-1008. doi:10.1093/logcom/exi041
- Swenson, K. (1993). A visual language to describe collaborative work. Visual Languages, 1993., Proceedings 1993 IEEE Symposium on (pp. 298-303). Bergen: IEEE.
- Varro, D. (2003). Towards Symbolic Analysis of Visual Modeling Languages. Electronic Notes in Theoretical Computer Science, 72(3), 51-64. doi:10.1016/S1571-0661(04)80611-X
- Visual Contract Language.
  Retrieved March 7, 2012, from http://vcl.gforge.uni.lu/
- Visual OCL.
  Retrieved March 6, 2012, from http://tfs.cs.tu-berlin.de/vocl/
- VENSIM
  Retrieved March 5, 2012, from http://www.vensim.com/
- AnyLogic
  Retrieved March 5, 2012, from http://www.xjtek.com/

Appendix B

Requirements

The following items are an excerpt from the *"Scenario requirement description document"* accessible at `http://tinyurl.com/cnqupjj`.

B.1 BPMN2 model requirements

B.1.1 Structural requirements

- BMS1 The model diagram, subsequently model, holds a Collaboration or Process.
- BMS2 The model has Events.
- BMS3 The model can have Activities.
- BPS4 The model can have Gateways.
- BPS5 The model can have Connections.
- BPS6 The model can have Swimlanes.
- BPS7 The model can have Artefacts.
- BPS8 Events are Start Events, End Events, or Intermediate Events.
- BPS9 Events have a mode and a type.
- BPS10 Activities are either Tasks or Processes.
- BPS11 Gateways have a type.
- BPS12 Connections are Sequence Flows, Message Flows, or Associations.
- BPS13 Swimlanes are either a Pool or a Lane.
- BPS14 Artefacts are Data Objects, Groups, or Annotations.
- BPS15 Pools can contain multiple Lanes.

B.1.2 Behavioural requirements

None. Behaviour is expressed in model instances only, using the model elements to express behaviour.

B.1.3 Constraints

- BMC1 Connections must have exactly one source and one target.

B.2 Stock Management model requirements

B.2.1 Structural requirements

- SS1 A customer has a portfolio, and a standing which can be either good or bad.
- SS2 Shares are either common stock or preferred shares.
- SS3 Preferred shares may be convertible in which case they are convertible preferred shares which have a conversion rate and an earliest conversion date.
- SS4 An order is placed to either buy or sell a given number of stocks.
- SS5 A portfolio belongs to a customer and holds all stock he owns.
- SS6 A customer has a portfolio at the stock brokerage.
- SS7 A broker works for the stock brokerage. He has a specialty which can be either as a seller or buyer of stock.
- SS8 A broker handles orders from customers.
- SS9 A customer holds credentials which are used to identify with the stock brokerage.

B.2.2 Behavioural requirements

- SB1 A customer can check his portfolio.
- SB2 A customer can issue an order to buy or sell stock.
- SB3 A customer can review his portfolio.
- SB4 A customer can wait for his order to be fulfilled.

B.2.3 Constraints

- SC1 A customer cannot sell more shares than are in his portfolio.
- SC2 A customer in bad standing cannot buy preferred stock.
- SC3 A customer can only login with the correct credentials.
- SC4 A customer has at least one portfolio.

- SC5 Only a broker with the right specialisation can handle an order.
- SC6 A customer in bad standing cannot have more than two portfolios.

B.3 Tangible User Interface model requirements

B.3.1 Structural requirements

- TES1 A widget has one or more handles.
- TES2 A handle can be bound or unbound.
- TES3 A handle has a continuous attribute, Domain Presence in the range]0;1].

B.3.2 Behavioural requirements

- TEB1 Dropping a handle on the table may bind the widget the handle is associated with to the underlying.
- TEB2 Lifting the handle removes the Domain Presence of said handle.
- TEB3 Shaking clears the mode of all widgets whose mode is clearable.
- TEB4 Shaking removes the anchored entity from widgets whose mode is not clearable.

B.3.3 Constraints

- TEC1 A widget has at least one handle.

B.4 Tangible User Interface scenario requirements

B.4.1 Structural requirements

- TMS1 A widget holds an ID, a position, rotation angle, and clearable state.
- TMS2 A widget may hold a visualisation.
- TMS3 Widgets can be differentiated by their mode.
- TMS4 A widget can have a state.

B.4.2 Behavioural requirements

- TMB1 A widget can be activated.
- TMB2 A widget can be rotated.
- TMB3 A widget can be dragged or slid over the canvas.
- TMB4 Rotating a widget in the Zoom mode zooms the viewport in.

- TMB5 Rotating a widget in the View mode rotates the viewport left.
- TMB6 Rotating a widget in the Expander mode rotates the attached visualisation left.
- TMB7 Moving two Spreader widgets apart increases the canvas space in between them.
- TMB8 Moving two Spreader widgets together decreases the canvas space between them.
- TMB9 Sliding a widget in the Sword mode across all of model entity deletes it.
- TMB10 Sliding a widget in the Sword mode over parts of a model entity clears its contents.
- TMB11 Moving a widget in Hand of God mode moves the grabbed component(s) as well.
- TMB12 Activating a widget on the canvas depends on the widgets mode and the underlying canvas area. Consult table B.1.
- TMB13 Dropping the Stamp on the canvas creates the model entity the Stamp is currently imprinted with.

B.4.3 Constraints

- TMC1 The rotation angle of a widget in the Spreader mode is modular to 45.
- TMC2 The enclosed space drawn from a Lasso must not cut across canvas sub-areas.

Canvas area	Widget mode	Desired Behaviour
Free space	Knob	Anchor the widget to enable the sliding of the viewport.
	Hand of God	Release the previously grabbed entity.
	Lasso	Draw an enclosed space creates a canvas sub-area.
	Chain	Show radial menu to select target component.
	Clone	Place copy of the component(s) grabbed by the first handle.
	Teleporter	Cut the component(s) grabbed by the first handle and place them at the second handles location.
Toolbox - Model entities	Stamp	Switch the stamp to the underlying model entity.
	Clone	Change the type of the model entity grabbed by the first handle in respect to the underlying toolbox model entity.
Toolbox - Modelling aids	Any	Attribute a new mode.
Model entity	Chain	Show radial menu to select connector or component.
	Hand of God	Grab the underlying model entity.
	Clone	Grab the underlying model entity.
	Teleporter	Grab the underlying model entity.
Any	View	Reset viewport to default.
	Expander	Hide respectively show sub-process.
	Wormhole	Anchors the wormhole end-point to the enclosing canvas area.

Table B.1: Table of behavioural requirements depending on activation zone

Appendix C

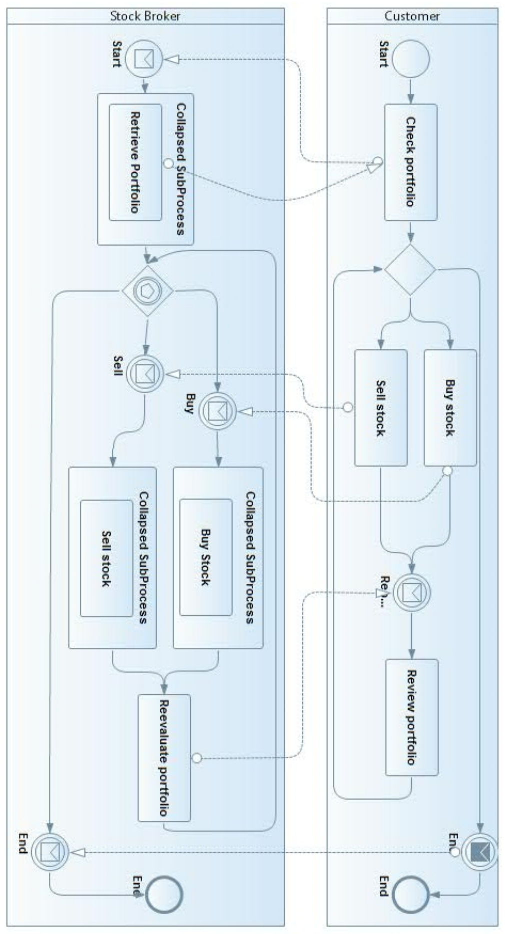


Figure C.1: BPMN2 scenario instance at second level of refinement.

Appendix D

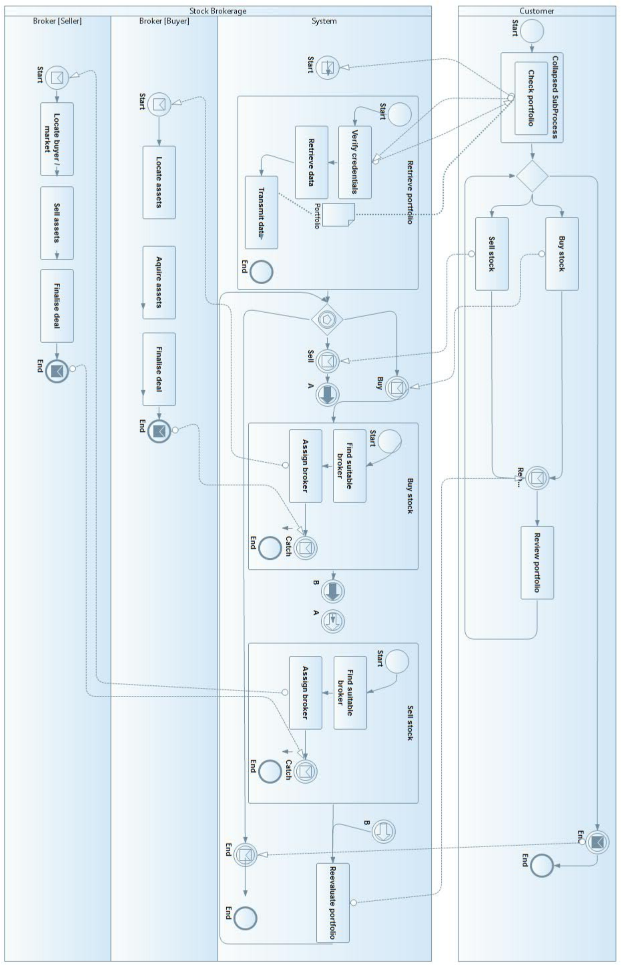


Figure D.1: BPMN2 scenario instance at third level of refinement.

Appendix E

Weighting evaluation criteria

To find a Visual Modelling Language (VML), such as UML, suitable to express the modelling of Business Process Model and Notation 2 (BPMN2) models using a Tangible User Interface (TUI), several criteria have been distilled. It is important to weigh these criteria carefully and gather as much feedback as possible. Therefore, this small study has been devised to gauge the user perspective.

The following six broad criteria are used to judge the suitability of VML. Please rate how important it is for you that a VML you would want to use fulfils the criterion?

	Not important at all	Somewhat important	Neutral	Important	Very important
Tool support					
Semantics & Transformation					
Expressivity					
Usability					
Error checking					
Validation & Verification					