The University of Auckland
Library Thesis Consent Form

This thesis may be consulted for the purposes of research or private study provided that due acknowledgement is made where appropriate and that the author's permission is obtained before any material from the thesis is published.

Author of thesis	Fan Zhang
Title of thesis	Support for Dynamic Supply Chain Modelling
Name of degree	Master of Science in Computer Science
Date	April 26, 2010

Please tick the boxes that apply

Print Format

☐ I agree that the University of Auckland Library may make a copy of this thesis available for the collection of another library on request from that library.

☐ I agree to this thesis being photocopied for supply to any person in accordance with the provisions of Section 56 of the Copyright Act 1994.

Digital Format

I certify that the digital copy of my thesis deposited with the University will be the same as the final officially approved version of my thesis. Except in the circumstances set out below, no emendation of content has occurred and I recognise that minor variations in formatting may occur as a result of the conversion to digital format.

☐ I confirm that my thesis does not contain material for which the copyright belongs to a third party.

or

☐ I confirm that for all third party copyright material in my thesis I have either:

a) obtained written permission to use the material and attach copies of each permission

or

b) removed the material from the digital copy of the thesis; fully referenced the deleted materials and, where possible, provided links to electronic sources of the material.

Signature of Author: _____ Date: _____

Comments on access conditions

Graduate Centre only: Digital copy accepted ☐ Signature Date

Support for Dynamic Supply Chain Modelling

Fan Zhang

This thesis is submitted in fulfilment of the requirements for the degree of Master of Science in Computer Science, completed at The University of Auckland.

April 2010

Abstract

Dynamically forming the most optimized supply chain networks is one of the key factors for long term business success. In modern business world, Small-to-Medium Enterprises (SMEs) need to dynamically collaborate and coordinate to each other for the emerging business opportunities. It is increasingly challenging for SMEs to compete successfully without well-designed and evolvable supply chain networks. Many approaches aim to support the modelling of such networks. One example is the EC-funded project "SMEs Undertaking Design of Dynamic Ecosystem Networks" (SUDDEN), which provides an infrastructure for value-added process coordination and supply chain management for SMEs.

Unfortunately, due to the complexity of most supply chain models, non-specialist end users find them difficult to understand, build and reconfigure. This motivates us to provide support for them to participate in complex supply chain network modelling and enable them to share their domain specific knowledge. Challenges for our research include (i) a lack of generally accepted DSVL design and evaluation techniques to help ensure a "good" solution; (ii) the research team for our project being highly distributed (UK and New Zealand, with our end users in Austria); and (iii) the need to iteratively develop both the DSVLs and their support tool while gaining end user feedback.

In this thesis, we present a suite of domain-specific visual modelling languages (DSVLs) and a suitable support tool by using the SUDDEN concept model to assist SMEs in supply chain modelling. We present a new approach for the design of our visual languages, which leads to our research on scientific visual notation design. We also present our collaborative development process across our distributed research team. Finally, we describe our experiences with these and the feedback we obtained from our target end users on our modelling languages and support tool.

Acknowledgements

I would like to express my genuine thanks and appreciation to my supervisors, Prof John Hosking and Prof John Grundy, for their precious guidance and support. Without their knowledge, commitment and mentor, I could not achieve my goals for my research. I believe that I will benefit from their insight, talent, and experience for the rest of my life.

Secondly, I would like to express my appreciation to Dr. Nikolay Mehandjiev and the rest of the SUDDEN team for their invaluable contribution, advice and feedback to my work; otherwise it is not possible to achieve such quality work.

Also, I would like to extend my thanks to my entire family, especially my parents and my wife, for encouragement and support during my study. They made my life so beautiful and enjoyable.

Last but not least, I want to thank the nameless souls for their time and effort spent for helping me out in my informal user evaluation and empirical software engineering study.

Table of Contents

Abstract ii
Acknowledgements iii
List of Figures ix
List of Tables xi
Chapter 1. Introduction 1
1.1. Motivation 2
1.2. Goals and Objectives 2
1.3. Methodology 2
1.4. Thesis Overview 4
1.5. Summary 4
Chapter 2. Background and Related Research 6
2.1. Knowledge Management 6
2.1.1 Knowledge Management (KM) Overview 6
2.1.2 Types of KM systems 7
2.1.3 Semantic Web based KM systems 7
2.1.4 Knowledge Life Cycle (KLC) in Semantic Web based KMS 8
2.1.5 Using Ontologies for KM 9
2.2. Supply Chain Modelling 9
2.2.1 Supply Chain Overview 9
2.2.2 Supply Chain Design and Management 10
2.2.3 Supply Chain Models 11
2.2.4 The Supply Chain Operations Reference-model (SCOR) 13
2.3. Visual Languages (VLs) 13
2.3.1 Visual Languages Overview 13
2.3.2 Visual Language (VL) Design 14
2.3.2.1. The Cognitive Dimensions (CDs) framework 14
2.3.2.2. The Physics of Notations framework (Moody, 2009a, 2009b) 15
2.3.3 Domain Specific Visual Language (DSVL) Overview 16
2.4. Agile Methodologies 17
2.4.1 Extreme Programming 17

2.4.1.1. The values 17
2.4.1.2. The principles 18
2.4.1.3. The XP practices 18
2.4.2 SCRUM 19
2.4.2.1. The SCRUM Lifecycle 20
2.4.2.2. The Principles of SCRUM 20
2.4.2.3. The SCRUM Master runs the SCRUM project 21
2.4.3 Distributed Agile and Distributed SCRUM 21
2.4.3.1. Challenges 21
2.4.3.2. Distributed Agile Practices 22
2.5. Model Driven Engineering (MDE) 22
2.5.1 MDE Overview 22
2.5.2 Model Driven Architecture (MDA) Overview 24
2.6. Related Research 26
2.6.1 Onto-SCM 26
2.6.2 Supply Chain Simulation Tool 28
2.6.3 Other related tools 31
2.7. Discussion 31
2.8. Summary 34
Chapter 3. Requirements and Specifications 35
3.1. Introduction 35
3.1.1 Roadmap 35
3.1.2 Goals and Objectives 38
3.1.3 Approach 38
3.1.4 Documenting Requirements 39
3.2. Understand the Domain Model 41
3.2.1 The Product subsystem 43
3.2.2 The Process subsystem 44
3.2.3 The Abstract Supply Network (ASN) subsystem 47
3.3. The MaramaSUDDEN Visual Language Requirements 48
3.3.1 The Meta-Model Modelling Requirements 48
3.3.2 The Visual Notation Modelling Requirements 49
3.4. The MaramaSUDDEN Visual Modelling Environment Requirements 51
3.4.1 Functional Requirements 51

3.4.1.1. The boundaries of our visual modelling environment 51
3.4.1.2. Actors 52
3.4.1.3. Use Case Modelling 52
3.4.2 Non-functional requirements 55
3.4.2.1. GUI design requirements 55
3.4.2.2. Usability Requirements 56
3.5. Summary 57
Chapter 4. Software Engineering 2.0 58
4.1. Introduction 58
4.2. Web2.0 Adds New Value 59
4.3. Distributed Agile Methodology 63
4.4. Challenges for classic agile practices in a new environment 63
4.5. Process and Practices 64
4.6. Discussion 68
4.7. Conclusion and Future Work 70
Chapter 5. Visual Language Design 72
5.1. Introduction 72
5.2. Challenges and a rough Design 73
5.2.1 Challenges 73
5.2.2 Design 74
5.3. Visual Language Design 74
5.3.1 Visual Metaphor Design Paradigm 74
5.4. The elements of the visual notation paradigm 75
5.4.1 Generic Geometric Shapes 75
5.4.2 Colours 76
5.4.3 Symbolic textures 78
5.4.4 Line 79
5.4.5 Text 81
5.5. MaramaSUDDEN Visual Notation Design 82
5.5.1 Notation design for the Product DSVL 83
5.5.2 Notation design for the Process DSVL 85
5.5.3 Notation design for the ASN DSVL 87
5.6. Meta-Model Design 88
5.7. Summary 91

Chapter 6. Modelling Environment Design 92
6.1. Introduction 92
6.2. The Preliminary Design 92
6.3. Modelling Environment Design 93
6.3.1 The Choice of Meta Tools 93
6.3.2 Architecture design 94
6.3.3 Design view types 97
6.3.3.1. The Product View 98
6.3.3.2. The Process View 99
6.3.3.3. The Abstract Supply Network (ASN) View 101
6.3.4 Interface Design 102
6.3.4.1. Modularization 102
6.3.4.2. Expansion – Hierarchical Abstraction Support 103
6.3.4.3. Simplification 104
6.3.4.4. Optimization 105
6.3.4.5. GUI Windows Support 106
6.3.5 Facilities Design 107
6.3.5.1. Real-time/non-real-time Editing Mechanism Support 107
6.3.5.2. Consistency Validation Mechanism 109
6.4. Function Design 111
6.5. Summary 114
Chapter 7. Support Tool Implementation 115
7.1. Introduction 115
7.2. Implementing Model Driven Engineering (MDE) Technologies in supply chain knowledge management 115
7.3. Implementation Choices 116
7.3.1 Implementing functionality by using Java 117
7.3.2 Selection of GUI technologies 118
7.3.3 Choice of Visual GUI editors 119
7.4. Experience of using the Marama tools 119
7.4.1 Implementing Meta-Models 121
7.4.2 Implementing Visual Notations 122
7.4.3 Implementing multiple view types 124
7.4.4 Overall experiences 125

7.5. Implementing multi-threading techniques for our tool 126
7.6. Implementing Knowledge Management by using Jena Framework 127
7.7. Summary 128
Chapter 8. Evaluation 129
8.1. Introduction 129
8.2. Cognitive Dimensions Based Visual Language Evolution 129
8.2.1 Consistency 130
8.2.2 Hidden Dependencies 131
8.2.3 Error proneness 133
8.2.4 Closeness of Mapping 134
8.2.5 Diffuseness/Terseness 135
8.2.6 Hard Mental Operations 136
8.2.7 Premature Commitment 137
8.2.8 Role-Expressiveness 138
8.2.9 Secondary Notation 138
8.2.10 Viscosity 139
8.2.11 Visibility and Juxtaposability 139
8.3. Progressive Evaluation 140
8.3.1 Informal User Testing and User Evaluation 140
8.3.2 Preparation and Plan for the Informal User Test 141
8.3.3 Questionnaire Preparation for the informal user test 142
8.3.4 Discussion 144
8.4. Summary 146
Chapter 9. Conclusion and Future Work 147
9.1. Introduction 147
9.2. Contributions 147
9.3. Future work 149
9.4. Summary 150
Reference 152
Appendix A – Use Cases 160

List of Figures

Figure 1. A Taxonomy for Supply Chain Models. From ("Supply chain modeling: past, present and future", by Min & Zhou, 2002, Comput. Ind. Eng., 43, p. 240.) 12
Figure 2. The visual interface of Onto-SCM. From ("Ontology-based semantic models for supply chain management", by Ye, et al., 2008, 2008, The International Journal of Advanced Manufacturing Technology, 37, pp. 1258) 26
Figure 3. The visual symbols of Onto-SCM. From ("Ontology-based semantic models for supply chain management", by Ye, et al., 2008, 2008, The International Journal of Advanced Manufacturing Technology, 37, pp. 1251.) 27
Figure 4. The structure of Onto-SCM. From ("Ontology-based semantic models for supply chain management", by Ye, et al., 2008, 2008, The International Journal of Advanced Manufacturing Technology, 37, pp. 1252) 28
Figure 5. The structure of the simulation tool. From ("Supply chain simulation modeling made easy: an innovative approach", by Cope, et al., 2007, Proceedings of the 39th conference on Winter simulation, Washington D.C, IEEE press.) 30
Figure 6. The Generic GUI Interface. From ("Supply chain simulation modelling made easy: an innovative approach", by Cope, et al., 2007, Proceedings of the 39th conference on Winter simulation, Washington D.C, IEEE press.) 30
Figure 7. The Roadmap of Chapter 3. 37
Figure 8: The syntax for our requirements engineering models. Adapted from ("UML 2.0 and the Unified Process: Practical Object-Oriented Analysis and Design " (P.52), by Arlow & Neustadt, 2005, 2005, Addison-Wesley Professional.) 40
Figure 9: The key concepts and mechanisms of the domain model. From ("Recursive Construction and Evolution of Collaborative Business Processes" Mehandjiev, et al, 2009, p.3.) 41
Figure 10. The conceptual model for the Product sub-system. (This drawing is credited to Martin Carpenter, from the SUDDEN project, 2009.) 43
Figure 11. Dependencies between processes. From (“Coordination Models and User Interaction Language” by Carpenter,et al., The SUDDEN project Deliverable D2.1, 2007, p.16.) 45
Figure 12. Relationships of Actor, Goal and Process of the domain model. From ("Recursive Construction and Evolution of Collaborative Business Processes" Mehandjiev, et al, 2009, p.6.) 46
Figure 13. An example of ASN formation. 47
Figure 14: The use case diagram for our visual modelling environment. 53
Figure 15. Examples of Web2.0 applications used in our project. 61
Figure 16. Various tools and their uses in our GSD context 62
Figure 17. Our development process. 65
Figure 18. Generic shapes selected for our visual notation design paradigm 76
Figure 19. Theme colours selected for our design paradigm 78
Figure 20. Texture patterns used for “LogisticService”, “Node” and “Machine” to make these symbols more distinguishable. 79
Figure 21. Visual representations for relationships 80

Figure 22. Dotted lines used in our visual notation design. 80
Figure 23. A symbol for association types used in our design. 81
Figure 24. A screen shot for an example of the Product DSVL. 83
Figure 25. A screen shot for the Process DSVL. 85
Figure 26. A screen shot for the ASN DSVL. 87
Figure 27. An entity class of the MaramaSUDDEN meta-model. 88
Figure 28. Examples of entity relationships in the MaramaSUDDEN meta-models. 89
Figure 29. Multiple Entity Relationships: Between Part and Complex Part classes, there are two different relationships 89
Figure 30. The Meta-models of MaramaSUDDEN 90
Figure 31. The structure of the Marama meta- tools. 94
Figure 32. The four layers of the MaramaSUDDEN tool. 95
Figure 33.The Architecture of MaramaSUDDEN. 97
Figure 34. A simple product decomposition example in the Product view. 98
Figure 35. The three basic process dependencies from the MIT process handbook. 99
Figure 36. A generic example of goal driven process coordination. 100
Figure 37. An ASN forming example in the ASN view. 101
Figure 38. A set of instances (2) have been modulized into one symbol (1). 103
Figure 39. The hierarchical abstraction of the expansion technique. 104
Figure 40. The number of connectors in Palette is reduced to only one. 105
Figure 41. Line routers used to optimize ling crossing and minimize space waste. 106
Figure 42. A GUI control window to select existing instance names or create a new one for selected shape type. 107
Figure 43. The communication between the interface (top) and the database (bottom) can be synchronized or asynchronous. 108
Figure 44. A relationship between two instances (D and B) has a constraint to lock D o(left) from deleting. The drawing is credited to Jesús Héctor from the SUDDEN project. 110
Figure 45. Consistency Management. 111
Figure 46. An example of use case realisation for the “Select and Set Ontology” function. 112
Figure 47. A screenshot of MaramaSUDDEN under construction. 120
Figure 48. Visualised meta-model modelling by using the Marama Meta-model Definer. 122
Figure 49. The visual notations of MaramaSUDDEN under construction using the Marama Shape Designer. 123
Figure 50. The MaramaSUDDEN view types are under development. 125
Figure 51. An example of using multi-threading in MaramaSUDDEN. 127
Figure 52. Jena2 APIs used in MaramaSUDDEN to retrieve sub-resources of a resource. 128

List of Tables

Table 1. The Priority Attribute values for the documents of MaramaSUDDEN requirements. Adapted from ("UML 2.0 and the Unified Process: Practical Object-Oriented Analysis and Design " (P.59), by Arlow & Neustadt, 2005, 2005, Addison-Wesley Professional.).....40
Table 2. A Set of Use Cases for the MaramaSUDDEN Modelling Tool.....55
Table 3. User story examples from our backlog. 68
Table 4. A set of functions to assist users in the modelling environment.....114
Table 5. Closed Ended Questions for Our Information User Test.....143
Table 6. The Open-ended Questions of our Questionnaire.....144

Chapter 1. Introduction

Nowadays, the business environment is changing at an accelerating pace. To grasp emerging business opportunities and secure substantial business successes, Small-to-Medium Enterprises (SMEs) need to effectively collaborate and plan their business activities and processes, especially within complex supply chain networks. The EC-funded SMEs Undertaking Design of Dynamic Ecosystem Networks (SUDDEN) (Mehandjiev, Stalker, & Carpenter, 2009) project aims to assist SMEs in designing and coordinating with each other collaboratively in supply chain modeling in a innovative way. A conceptual model has been constructed by the SUDDEN project team. It is expected that a large number of supply chain practitioners will engage in supply chain modelling using this knowledge infrastructure (Mehandjiev, et al., 2009). The problem is how to assist these supply chain practitioners to use, manage, share domain-specific knowledge efficiently, and how to improve the semantic interoperability of knowledge.

Visualization is one of the key software engineering approaches that can be used to support software applications in many domains, e.g. CADs, CASE tools (Diehl, 2005). Visualisation can be used to improve the presentation of complex models to users, allow users to manipulate models in more natural ways, and provide knowledge and data presentations in much more readily understandable forms. In recent years, visualisation for knowledge management has been used in the supply chain domain to assist visual knowledge design, decision support and knowledge interoperability (Cope, Fayez, Mollaghasemi, & Kaylani, 2007) (Chatfield, Harrison, & Hayya, 2009; Tavassoli, Sardashti, & Toussi, 2009; Ye, Yang, Jiang, & Tong, 2008). In this thesis, we describe a suite of domain-specific visual languages and support tool using the SUDDEN model to support dynamic supply chain modelling. Our tool allows SME users to visually design and manipulate complex supply chain models and knowledge, and to share knowledge across a number of applications and organisations with the support of visual knowledge metaphors, e.g. business processes. To increase the usability of our tool, we worked on different aspects of usability, e.g. optimization and simplicity of the user interface. Also, we carried out a continuous evaluation for our tool development with active end-user involvement.

1.1. Motivation

Supply Chain modelling normally involves reusing existing domain knowledge, creating new knowledge, and sharing supply chain knowledge with other supply chain partners. To quickly design or customise models using exiting knowledge for different modeling scenarios, supply chain consultants face many technical and non-technical problems, e.g. effective reuse of existing knowledge models to solve problems. The SUDDEN project provides a theoretical framework to support knowledge sharing and knowledge transformation among supply chain partners. The SUDDEN model covers some basic supply chain concepts and knowledge, which are broadly used in supply chain modelling (Mehandjiev, et al., 2009). Our motivation for this research project was to integrate DSVLs and the SUDDEN knowledge framework into a visual supply chain modelling tool. The tool should provide an efficient and easy-to-use interface to facilitate SMEs users to visually solve supply chain problems, coordinate different business processes, and share/reuse domain knowledge.

1.2. Goals and Objectives

There are different approaches to support Supply Chain modelling for different purposes, e.g. decision support (Cope, et al., 2007). Our goal in this research is to help SME users to model supply chain networks visually, and share their supply chain knowledge semantically. More precisely, we have three key objectives to achieve to reach this goal. First of all, the prototype tool will experiment with the concepts of domain-specific knowledge visualisation by using both Green's CDs framework (Green & Petre, 1996) and Moody's Physics of Notations framework (Moody, 2009a) for improved visual notation design. Secondly, we need to develop a good software engineering based (e.g. MDA) modelling environment to realize our domain-specific visual languages. Finally, the visual environment needs to be easy-to- use and provide a rich set of functions for SME users.

1.3. Methodology

We started with a literature review which came across a number of related research areas. We focused on finding out the state of the art for supply chain modelling, knowledge

management, visual language design, agile software development, model driven engineering, and related work.

Based on our literature review, we outlined the scope of our research, identified the key research value for our work, and established a theoretical framework to carry out our research. We believed it is very important to get end users involved actively on the actual development. Hence, we set up the MaramaSUDDEN project with the SUDDEN team, which ensured different stakeholders that were engaged in our tool development.

In order to deliver the best possible result for our visual language design, we adopted both the CDs framework and the Physics of Notations framework. As the SUDDEN knowledge model was developing, it was getting more complex and comprehensive. There were several dozens of concepts and relationships which need to be visualised. It would be overwhelming for our target users if they have to manage a large number of visual notations within a limited computer screen space. Therefore, we adopted a multi-view and multi-editor approach (John Grundy, Hosking, Huh, & Li, 2008). Based on the key sub-systems of the SUDDEN model, we split our visual language tool interface into three views. Each of these views addresses a range of supply chain issues. For the modelling environment development, we adopted the Marama meta-tools which support MDE based visual language development (John Grundy, et al., 2008). Based on the users' requirements, we used a multiple-layer design, including an interface layer and a business logic layer and other two layers.

The research project team comprised the author, two researchers and a research associate in New Zealand, a researcher in the UK and two research associates in the UK. The author and UK research associates needed to work closely and collaboratively together for much of this project. To deliver our research results with the difficulties of distributed software development we adopted some agile (XP and SCRUM) practices for our project development. For example, we used an incremental iterative development method. We also incorporated users' feedback during the life of our project and carried out a continuing evaluation approach for our visual languages and prototype toolset.

1.4. Thesis Overview

- Chapter 1: Introduction – we introduced our research interests, summarised our motivation for choosing the topic, our key research goal and supporting objectives, and the methodology we used to achieve our goal.
- Chapter 2: Background and Related Research – we cover technologies, theories, concepts, techniques, background knowledge and related research that are relevant to our work.
- Chapter 3: Requirements and Specifications – we discuss the key requirements for our visual languages and their support tool based on the literature reviewed in the previous chapter and initial users' feedback.
- Chapter 4: Software Engineering2.0 – we discuss the agile method and Web2.0 used for our development process.
- Chapter 5: Visual Language Design – we present the details of our visual language design.
- Chapter 6: Tool Design – we describe the design decisions we made for our modelling environment.
- Chapter 7: Support Tool Implementation - we describe our support tool implementation and key decisions made when realising our visual modelling tool.
- Chapter 8: Evaluation – we present the evaluation for our DSVLs and support tool. We discuss the strengths and weaknesses of our tool focusing on the usability by using the cognitive dimensions framework and an informal user evaluation.
- Chapter 9: Conclusion – we summarise the main contributions of our research and identify key areas for future work.

1.5. Summary

In this chapter, we outlined our main research goal, our motivation for this research and our approach taken on this research project. We defined and discussed the scope of our research, which aims to provide a suite of DSVLs and tool support for domain-specific supply chain modelling and knowledge management. We put our research emphasis on constructing the visual knowledge representations by using both the CD and Physics of Notations frameworks, and development of an easy-to-use prototype modelling environment. Finally, we presented an overview of this thesis which provides a guide for all chapters for our readers.

The details of our work will be covered by the following chapters: Requirements and Specifications, Software Engineering 20, Visual Language Design, Tool Design, Implementation, and Evaluation.

Chapter 2. Background and Related Research

In this chapter, we present research related to our study. The purpose of this literature review is as follows:

- To understand the theories, knowledge and concepts related to our research (e.g. visual language design), and establish a theoretical environment for our research and development.
- To define key terminologies for related technologies and techniques, e.g. supply chain modelling.
- To identify capacities of related research, and define the scope of our work, then identify the research value of our work.

Our background and related research covers a number of areas, e.g. visual language design, knowledge management. Some of the published materials are fundamental concepts of our work, e.g. Moody's research (Moody, 2009a), and some of them gave us a better understanding of a particular aspect of our work. The related research covers some non-commercial supply chain modelling tools and solutions, which have been recently proposed by other researchers.

2.1. Knowledge Management

2.1.1 Knowledge Management (KM) Overview

Knowledge Management (KM) refers to the management of intelligent information and data (e.g. experience) are processed and preserved, which can create value for organisations or persons in a certain context (Alavi & Leidner, 2001). As information technologies are developing, many IT driven KM systems have been developed. Wilson & Snyder (1999) indicate KM should work in two ways to deliver intelligent data. Firstly, it should create a perceptive framework to define information with a certain rules. Secondly, it should deliver information to help users decide how to solve problems and achieve higher performance. Wilson & Snyder (1999) suggest that IT and KM need each other: that they can benefit from

each other. According to Wilson & Snyder (1999), IT-driven KM can enhance productivity and achievement in business environments.

Alavi and Leidner (2001) summarise KM issues into four categories: storage, creation, transfer and retrieval. Furthermore, according to Verwijs et al., (cited in Ribino, Oliveri, Re, & Gaglio, 2009), there are four common approaches to manage knowledge in commercial environments: knowledge storage, knowledge processes, learning processes , and intellectual capital. For knowledge storage, Verwijs et al., (cited in Ribino, et al., 2009) state that knowledge obtained and learnt (e.g. solutions) from both internal and outside resource needs to be standardized and preserved. For knowledge process, knowledge is used as an element to create value (e.g. providing solutions). The key point of this approach is how to use knowledge via personal involvement. The learning process is about association and interaction between personal and enterprise learning. KM is used to aid enterprise learning by capturing and storing new information. For intellectual capital, knowledge is treated as an intelligent asset for organizations, which aims to extract formalised knowledge to improve the different aspects of the enterprises, e.g. education (cited in Ribino, et al., 2009).

2.1.2 Types of KM systems

Ribino et al. (2009) classify the types of KM systems into three major categories: a document based KMS, an ontology based KMS, and an AI based KMS. For document based KM systems, a number of document types are used to share and preserve knowledge such as MS word. For ontology based, knowledge is organised into ontologies with a certain structure and a set of pre-defined terminologies. An ontology is one form of meta-model which is used to describe concepts and the relationships in certain domains. For AI based approaches, Ribino points out that the use of AI engines in KM systems aim to find out possible solutions for particular problems (Ribino, et al., 2009).

2.1.3 Semantic Web based KM systems

At present, many knowledge management systems have shifted onto web-based platforms, such as intranet, Wiki sites. Zaihisma Che & Abdullah (2008) predict “Connectivity and interoperability” will become the key focus regarding to the evolvement of web based KM

systems and Semantic Web technologies, which improve “heterogeneity and interoperability” of knowledge.

Davies et al. (2007) state that using Semantic Web technologies is a promising way to improve KM as knowledge can be processed by both machine and human with ease, and knowledge and intelligent information can be retrieved and integrated effectively. Also, Web based KM systems improve accessibility and visibility of KM (Davies, et al., 2007) . The processed knowledge in Semantic Web based systems is more intelligible and understandable by machines, which supports comprehensive reasoning. Davies et al. argue Semantic Web technologies can “empower and energize” KM systems at different levels (Davies, et al., 2007) .

2.1.4 Knowledge Life Cycle (KLC) in Semantic Web based KMS

Chao et al. (2009) outline the knowledge life cycle (KLC) in the Semantic Web. According to Chao et al., the KLC include six phases, which are representation, interconnection, reasoning, retrieving, validation and integration (Chao, et al., 2009) .

For knowledge representation, Chao et al. argues that normal web content is only for human users; however, the format of knowledge in the Semantic Web is designed for machines. In addition, knowledge in Semantic Web format allows computer systems to manage it in a meaningful way (Chao, et al., 2009). Chao et al. also point out the major formats of Semantic technologies are “Unicode, XML (Extensive mark-up language), RDF/RDFs (Resources Description Framework / Resources Description Framework Schema), and OWL (Web Ontology Language)” (Chao, et al., 2009).

For knowledge interconnection, Chao et al. indicate the use of Uniform Resource Identifier (URI) and Namespaces in the Semantic Web ensure information from different platforms can interact and communicate each other (Chao, et al., 2009).

For knowledge reasoning, Chao et al. say that it is hard to search for the correct information semantically in the traditional web. However, it is possible to reason about knowledge in a semantic way, since the knowledge in the Semantic Web is powered by ”meta-data and rules” (Chao, et al., 2009).

For knowledge validation, to avoid invalid data, Chao et al. state that information retrieved needs to be validated. Chao et al. suggest checking and validating knowledge against pre-defined rules by using Rules Languages (e.g. SWRL). The validation can be "based on authenticity and integrity" (Chao, et al., 2009).

For knowledge integration, Chao et al. (2009) maintain knowledge can be integrated in three ways: putting knowledge together by techniques like MOM; incorporating knowledge as functions by using APIs; or adapt knowledge to business process.

2.1.5 Using Ontologies for KM

Varma (2007) says the primary reason for using ontologies in KM is because ontologies are platform independent, which improves knowledge sharing among different systems, and predefined vocabulary (e.g. relationships) enables knowledge to be searched in semantic way (Varma, 2007). Moreover, Varma (2007) states that ontology technologies assist KM in different ways, e.g. knowledge repositories and/or pre-defined vocabulary framework. Varma (2007) indicates that there are two major types of knowledge management: "content staging" and "content delivery". Ontology technologies can assist KM in both areas. Furthermore, ontology also improves the capacities of KM, in terms of sharing, interconnection, maximization of knowledge retrieval and reuse and reasoning support (Varma, 2007).

2.2. Supply Chain Modelling

2.2.1 Supply Chain Overview

A supply chain is a distribution network which uses resources to relocate resources (J. Wu, Ulieru, Cobzaru, & Norrie, 2000). It can also involve transforming one kind of resource to another with new value added services. Many people use logistics and supply chain network interchangeably. In fact, they are different. A supply chain refers to businesses processes involving many businesses. In contrast, logistics only means the processes inside a single enterprise. A supply chain comprises many logistical activities from different businesses (J. Wu, et al., 2000).

Supply chains can be small like a single company or could be very complex, e.g. an automobile supply chain can involve thousands of partners and enterprises. In the modern business world, Supply Chains act like high speed motor ways which interconnect suppliers and consumers. Each enterprise works like a car or a gas station on the "supply chain motorways". They deliver a range of services or products to each other. Supply Chains are the backbone for enterprises to stay successful. An effective Supply Chain is one of the critical elements for organisations to win business opportunities (J. Wu, et al., 2000).

2.2.2 Supply Chain Design and Management

In the past, different business processes (e.g. buying, transportation, inventory control and manufacturing) ran separately, which led to a huge amount of resource waste (Dangelmaier, Heidenreich, & Pape, 2005). Each of these processes had different goals and different planswhich did not coordinate with each other. Since the goals of these processes are different, business processes often conflicted (Dangelmaier, et al., 2005). Consequently, the processes were often delayed. Furthermore, due to markets changing faster, businesses face many challenges to deliver their products or services effectively and efficiently. Therefore, Supply Chain Management (SCM) was introduced. SCM is capable of integrating business requirements and resources from different business activities and processes to deliver best outputs for one single unified business object (Dangelmaier, et al., 2005).

Nowadays, with the introduction of information technology (IT) based collaboration, it is very common that businesses and enterprises use IT based supply chain management systems to organize and plan different processes within their supply chain networks. The idea of supply chain network is that a set of businesses, logistical facilities, physical sites, resources and systems work together to transport and transform the resources and services to meet customers' demands (Tavassoli, et al., 2009).

As technology has developed, many new information technologies have been introduced to enhance the capacity of supply chain systems, e.g. inventory control systems and Enterprise Resource Planning (ERP) systems. The introduction of new technologies advanced supply chain networks in many ways (Tavassoli, et al., 2009). For instance, with the support of RFID, businesses within a supply chain network can locate and track how many products

there are and where they are located at any time anywhere (Campos & Zorzo, 2007). IT provides a more effective and efficient way to manage stock level and productivity (Campos & Zorzo, 2007).

The major benefit of using Supply Chain Management and technologies is to maximise and optimize the capacities of the supply chain networks and reduce unnecessary resource waste (Tavassoli, et al., 2009). Within the supply chain networks, many goals and plans become one unified business objective and every process is under one single plan. All processes need to be planned according to the major goal. For example, lower inventory or faster distribution (J. Wu, et al., 2000). All processes are integrated and work according to the major goals. Supply chain management can not only help enterprises to achieve the goal, but also optimize the supply chain network (J. Wu, et al., 2000). The concept of SCM is not only to plan and design the processes within a single business or enterprise, but also to manage processes across different industries and manufacturers (J. Wu, et al., 2000). Based on different market requirements, businesses need to cooperate with each other within a supply chain in terms of human resource, information and technology resource, and other natural resources (Tavassoli, et al., 2009; J. Wu, et al., 2000).

2.2.3 Supply Chain Models

Since a typical supply chain involves many resources, e.g. information, enterprises and inventory (J. Wu, et al., 2000), Supply Chain Management faces many challenges. Since different industry or different businesses have different problem domains, one simple supply chain cannot solve all aspects of issues. At present, there are many supply chain models and approaches, which emphasise handling different problems of supply chains (Jiang & Tianfield, 2006).

Since supply chain management is a very wide research area, Min and Zhou (2002) categorize supply chain models into four key categories: deterministic models, stochastic models, hybrid models and IT-driven models. In their research, Min and Zhou classify the first two models (deterministic and stochastic) in a classic way (2002). In addition, there are two other models. One is the hybrid model (Bunick, Cited in Min & Zhou, 2002). Min and Zhou (2002) also suggest the reason to have a separated IT-driven model which identifies the

popularity and the power of information technology for supply chain models. Figure 1 indicates the taxonomy of supply chain models (Min & Zhou, 2002).

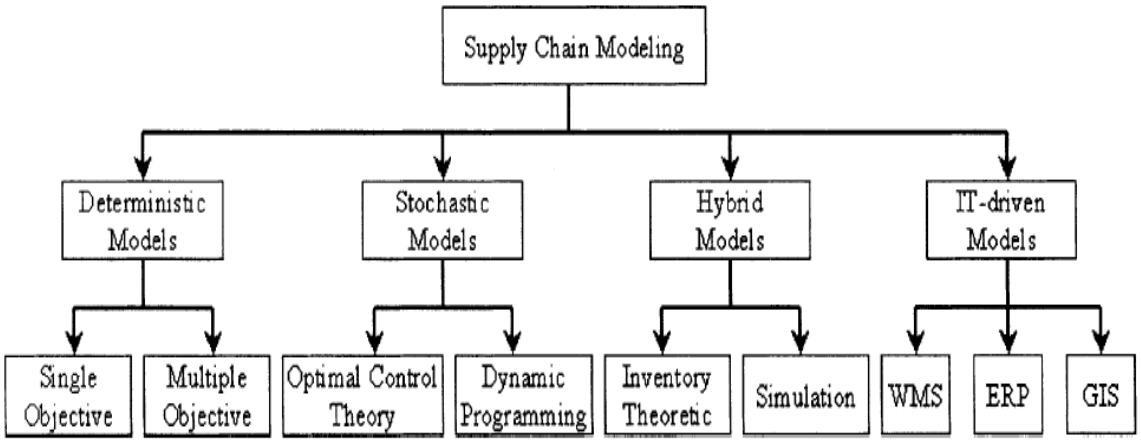

Figure 1. A Taxonomy for Supply Chain Models. From ("Supply chain modeling: past, present and future", by Min & Zhou, 2002, Comput. Ind. Eng., 43, p. 240.)

Min and Zhou (2002) divide the four major categories into sub-categories. They point out that the deterministic models include identified attributes and character (Min & Zhou, 2002). In contrast, the attributes of the stochastic models are unknown. Deterministic models are split into two sub-categories: "single objective" and "multiple objective models" (Min & Zhou, 2002). Min & Zhou (2002) classify stochastic models into optimal control models and dynamic programming models. Hybrid models include features and characteristics of both the deterministic models and the stochastic models. They are further split into inventory theoretic models and simulation models. They highlight the influence of IT for supply chain modelling. They classify IT models into three sub-categories: WMS, ERP and GIS models based on Shapiro's work (Cited in Min & Zhou, 2002). Furthermore, based on their purpose, Min & Zhou (2002) summarise these models into five major categories: supply selection and inventory control, production and inventory, location and inventory control, location and routing, and inventory control and transportation. Min & Zhou (2002) also identify a few major goals for supply chain modelling: customer service, monetary value, information/knowledge transactions, and risks.

Maurizio & Mariagrazia (2008) suggest there is a trend to decentralized models with internationalisation of global markets and they highlight growing market demands for modelling and coordination tools.

2.2.4 The Supply Chain Operations Reference-model (SCOR)

The SCOR model is the industry standard reference model (Council, 2008). The SCOR model is one of the most popular supply chain models and was introduced by the Supply Chain Council (Cope, et al., 2007). The SCOR model refers to Supply Chain Operations Reference model (Council, 2008). The SCOR model identifies five major Plans: Source, Make, Deliver and Return (Council, 2008). The users of our tool need to use this model as a template and customise it for specific requirements and demands. There are many supply chain works which are based on the SCOR model (Changrui, Jin, Hongwei, & Wei, 2006; Hai-Ying, Stucky, & Yan-Bin, 2008; Jin, Hongwei, Changrui, & Wei, 2006; Kasi, 2005).

2.3. Visual Languages (VLs)

2.3.1 Visual Languages Overview

There are many kinds of language systems, e.g. textual languages, visual languages (Moody, 2009b). Most advanced natural languages have both verbal and written symbols and characters with different rules. Visual Languages (VL) are not new to human societies. Human beings have been using visual languages for thousands years (Horn, 1999) For software engineering, visualisation is one of most important aspects of modelling and understanding software (N. Hari Narayanan & Roland, 1998). Researchers have proposed a number of tools and approaches to facilitate visualisation of software development and software design. For example, the Marama meta-tools are used for rapid VL generation (John Grundy, et al., 2008). Computer aided visual languages normally comprise a set of pre-defined visual notations with certain rules, including visual syntax and visual semantics (*Visual language theory*, 1998), e.g. UML or Microsoft Excel. In addition, it is common for visual language environments to support different level of code generation, which effectively improves the productivity of software development (Zhang, Zhang, & Cao, 2001). Traditionally, productivity of software development always depends on the software developer's capacities and experience (Banker, Datar, & Kemerer, 1991).

Nowadays, using visual languages and support tools (e.g. CASE tools) for software architecture, analysis and design, has become common practice in both industry and research communities. Moody (2009a) suggests that the visual icons and metaphors are easier to be

understood and remembered by human than other forms e.g. text. Visualisation is a more effective and efficient way for human beings to communicate with computers than text (Horn, 1999). However, it is difficult to design a good visual language. A number of theories and frameworks have been developed to support visual language design and development, for instance, the Cognitive Dimensions (CDs) framework (Green & Petre, 1996) and the Physics of Notations framework (Moody, 2009a). We discuss both of them in the following section.

2.3.2 Visual Language (VL) Design

2.3.2.1. The Cognitive Dimensions (CDs) framework

According to Blackwell (2008), the CDs framework (Green & Petre, 1996) has been used for analysis and evaluation for VLs for many years (cited in Moody, 2009b). Blackwell (2008) states "CDs framework was specifically developed to provide better guidance for those hoping to create notations with improved usability". Blackwell summarises the contributions of the CDs framework for visual languages design (2008). Firstly, the purpose of developing the CDs framework was not only for visual language design, but also for providing a conceptual platform for visual language designers to manage visual metaphors. Secondly, Blackwell states that the CDs framework can be used to emphasize "design trade-offs" (2008). In other words, gaining capabilities in one dimension of VLs could cause losing capabilities in other dimensions. Thirdly, Blackwell (2008) highlights the CDs framework's capabilities in usability analysis, specifically useful for "user interface operation". In addition, Blackwell (2008) notes the CDs framework focus on how users can employ visual notations in unexpected manners. The CDs framework has also been developed and used for number of VL design practices. For instance, Yang, DeKoven, & Zloof, (1996) used the CDs framework to develop a set of benchmarks for visual programming design .

Green & Petre (1996) state that the CDs framework provides a "Psychological**"** view for visual language designers, which addresses HCI issues mostly. The dimensions used for usability discussions include Abstraction Gradient, Closeness of mapping, Diffuseness, Error – proneness, Hidden dependencies, Progressive evaluation, Role-Expressiveness, Secondary notation, Viscosity and Visibility (Green, 1989).

However, Moody (2009b) criticises the CDs framework as it "does not provide a suitable theoretical foundation for VL research." Moody argues that the Physics of Notations frame is more suitable to support visual language design. Firstly, Moody (2009a) maintains that the CDs framework is unscientific. Secondly, Moody (2009a) argues the CDs framework "was not specifically developed for VL".

2.3.2.2. The Physics of Notations framework (Moody, 2009a, 2009b)

Moody describes the Physics of Notations framework as an alternative approach to the classic CDs framework. Moody (2009a) introduces three major concepts used in the Physics of Notations framework: dependent variables, a descriptive theory and a prescriptive theory.

For the dependent variables, Moody states the effectiveness of users' cognitive experience is critical for analysis and evaluation of the visual metaphor design. The effectiveness of the visual metaphors should act as the goal of the visual notations design, which should allow users to measure them by using empirical studies (Moody, 2009a).

Moody (2009a) explains that descriptive theory is about how visual metaphors communicate and interact to each other. Moody indicates the visual notations need to communicate with each other in terms of "graphic design, visual perception and cognition". Moody suggests the visual notations with good communication design is more effective for conveying information (Moody, 2009a).

For the Prescriptive Theory, Moody describes "a set of principles for designing cognitively effective visual notations" (Moody, 2009a) which are based on the actual visual representation studies. According to Gregor and Jones (Cited in Moody, 2009a), the principles of this theory includes "scope and purpose", "constructs", "principles of form" and "function", "testable propositions", "justificatory knowledge", "principles of implementation", and "expository instantiation". Moody adds "artefact mutability" into the principle sets of Physics of Notations (Moody, 2009a).

Compared with the CD framework, Moody (2009b) highlights three advantages to use of the Physics of Notations framework for VL design: it is specifically designed for visual language research domain rather than any IT artefacts; it is falsifiable which means the provisional results of using the Physics of Notations can be tested by empirical study; and it comprises

both a theory for explaining and predicting and a theory for design and action, "which is a higher evolutionary form than the CDs framework".

2.3.3 Domain Specific Visual Language (DSVL) Overview

Deursen, Klint, & Visser (2000) describe a Domain Specific Language (DSL) as "a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain". According to Ross (Cited in Mernik, Heering, & Sloane, 2005), the concept of a DSL was developed in the early days of computer programming and was described as "application-oriented" (Sammet cited in Mernik, et al., 2005). So far, the motivation to develop a DSL is still the same, which is "an improved abstraction of the problem allowing for the rapid creation and maintenance of a complex application" (Sprinkle & Karsai, 2004).

Recently, DSLs have become more popular because of domain specific modelling (DSM), which is playing an important role in software engineering (Deursen, et al., 2000). DSM is a new software architecture and design approach for better productivity, easier reuse and maintenance (Deursen, et al., 2000). DSLs facilitate higher-level abstraction, and are specially designed languages to solve a certain range of problems for a particular domain (Mernik, et al., 2005), which is the opposite to general purpose languages. DSLs provide a communication environment for software developers and domain experts in more convenient ways. However, it is very difficult to develop a good DSL since it requires extended domain knowledge and capacities of software language development. The Domain-specific models can be specified by employing these support tools. Some DSL tools also support code generation (Mernik, et al., 2005).

A DSVL is a special kind of Visual Language, which integrates technologies and techniques from both DSLs and VLs (Sprinkle & Karsai, 2004). A DSVL uses visual metaphors and graphic notations to define and specify a limited domain (*Visual language theory*, 1998).

Since modern computers were invented, developers have sought approaches for communication between computers and human users, which has been most powerful motivation for develop visual interfaces/forms (Horn, 1999) .

Visualization is one of key components to DSVLs. Selker & Koved (1988) summarise a number of important visual elements, which includes visual alphabet, visual syntax, intercalation and structure and coverage. Brown (1997) states how to meet the requirements of graphical notations. Moody (2009a) suggests that human nerve systems have to be taken into the consideration and "diagrams can convey information more concisely". Much work has been carried out to maximise the usage and construction of DSVLs, e.g. Bottoni & Grau (2004), (John Hosking, Mehandjiev, & Grundy, 2008).

2.4. Agile Methodologies

The "agile" approach is currently one of the more popular software development methodologies in the software industry. The term "agile" covers a range of approaches and practices: Extreme Programming (XP), Scrum, the crystal methods, feature driven, and dynamic systems development (D. Cohen, Lindvall, & Costa, 2004) (Maurer & Melnik, 2006). Unlike traditional plan-focus methods, the agile approach focuses on creativity, productivity , collaboration and feedback (Moe & Aurum, 2008).

2.4.1 Extreme Programming

Extreme Programming (XP) places emphasis on the practices of the software development.

2.4.1.1. The values

According to Newkirk (2002), there are four key values for XP: communication, simplicity, feedback and courage. Newkirk (2002) states poor communication often leads to unsuccessful results. Among the factors for project development, communication is very important. An effective communication should involve all the stakeholders of the project. For simplicity, Newkirk (2002) says the development of the project should only meet the current requirements, and should not assume any requirements in the future. Since there are too many uncertainties in the future, the wrong decisions made could be very costly. Newkirt (2002) points out that the usage of customers' feedback in XP is very important. Each iteration should be driven by frequent and instant customer feedback. The feedback ensures continuous improvement. Practitioners should have enough confidence to make decisions and have courage to develop successful artefacts (Newkirk, 2002).

2.4.1.2. The principles

As identified by Lippert et al. (2002), XP has a number of important principles to follow: rapid feedback, assume simplicity, incremental change, embrace change, quality work, teach learning, small initial investment, play to win, concrete experiment, open and honest communication, work with people's instincts, accept responsibility, local adaptations, travel light and honest measurement. Lippert at al. point out the first five of them are the most important ones (Lippert, et al., 2002).

For rapid feedback, Lippert et al. (2002) suggest the sooner feedback is obtained, the better to acquire correct decisions. Based on psychological studies, the shorter the period, the closer to the right path (Lippert, et al., 2002). Needless or overtime thinking should be avoided in XP, as they normally lead to negative results. For simplicity, Lippert et al. summarise that XP practitioners can benefit several ways (Lippert, et al., 2002). For example, it is easier to implement a simple design. Consequently, feedback can be obtained sooner and faster. Also, simple designs can be modified and updated with ease. They also mention there is no need to assume requirements for the future, as these could be wrong. For incremental change, according to Lippert et al., the commitment of the development should be achievable (Lippert, et al., 2002). Furthermore, for large and complicated projects, there will be unpredictable mistakes. Small incremental changes will help developers to correct these errors more readily. For embracing change, developers need to fully support changes and they should understand the benefits of the changes. However, developers should not be forced to make changes. For quality work, customers need to set up the bottom line of satisfaction for developers' work. The feedback and customers' satisfaction are guidelines for developers to deliver quality work (Lippert, et al., 2002). For more details, please refer to Lippert, et al.'s work.

2.4.1.3. The XP practices

Newkirk (2002) summaries a range of key practices of XP: planning, small releases, metaphor, simple design, testing, refactoring, pair programming, collective ownership, continuous integration, 40-hour weeks, on-site customer, and coding standards . Newkirk (2002) calls these practices "the lifeblood" of XP. Newkirk also defines and describes the details of these practices as follows.

For planning, XP practitioners need to decide the range of project and tasks for each development cycle. The estimated commitments require input from different stakeholders. The tasks need to be prioritised accordingly (Newkirk, 2002). For small releases, Newkirk (2002) points out that development should shift into implementation rapidly, and use customers' feedback as input to improve the artefact. For metaphor, practitioners need to know behaviours of the systems. All stakeholders should be able to understand and communicate the metaphors (Newkirk, 2002). For simple design, Newkirk (2002) states that it is critical to avoid complexity for the design of the artefact. The design should only meet the current requirements. The potential development should not be considered or planned. As a result of using simple design, it is easier and faster for practitioners to implement. For testing, Newkirk (2002) suggests testing is a critical practice. Feedback of tests can measure the maturity of the artefact. Tests should be conducted by both developers and users. For refactoring, the XP practitioners should adjust their design of the artefacts according to the results of the evaluation and test. The modification of the artefacts should not affect the functions of the products. Newkirk (2002) indicates that refactoring should be a daily practice for developers. For pair programming, Newkirk (2002) comments that the two programmers should work together to produce source code. Pair programming is set to improve developmer communication and productivity. For collective ownership, Newkirk (2002) says any stakeholder should be able to modify the artefact. For continuous integration, according to Newkirk, (2002) artefacts can be upgraded frequently on a daily basis. For 40-hour weeks, Newkirk (2002) suggests XP practitioners should work until exhausted. But he also mentioned the negative impact for overtime working. For on-site customer, Newkirk (2002) states the customers should be recognized as a part of the development group. On-site customers are especially valuable for generating acceptance tests. For coding standards, Newkirk (2002) highlights communication is critical for keeping the source code uniformed and standardized. For more details, please refer to Newkirk's work.

2.4.2 SCRUM

The SCRUM approach highlights the project management of agile software development (Schwaber, 2004). SCRUM focuses on dynamic adaption of the development process. Most published SCRUM case studies have focussed on small teams (Moe & Aurum, 2008;

Paasivaara, Durasiewicz, & Lassenius, 2009; Srinivasan & Lundqvist, 2009), however, there are some cases where SCRUM has been adopted for large scale distributed projects (Paasivaara, Durasiewicz, & Lassenius, 2008).

2.4.2.1. The SCRUM Lifecycle

Cohen et al. (2004) summarise the terminologies and practices for the lifecycle of SCRUM. For pre-sprint planning, Cohen et al. (2004) state that the traditional planning documentation is referred to as a "release backlog" in SCRUM. The specific assignments are chosen, prioritized and organized in the sprint backlog. The tasks for the next sprint should be achievable, and they need to be done on time. The pre-sprint aims to figure out the scope and overview of the development, and schedule targets for the programmers (D. Cohen, et al., 2004).

A "sprint" means a development iteration in SCRUM (D. Cohen, et al., 2004). Once the assignments for the sprint are fixed, they should not be altered. Cohen et al. highlight that the daily SCRUM meeting is an important element to achieve the goals (D. Cohen, et al., 2004). The daily SCRUM meeting increases interaction and communication among the SCRUM team members. During a SRUM meeting, progress and problems can be reported and discussed. All team members share the same knowledge and information of the development and understand the challenges and targets (D. Cohen, et al., 2004).

The post-sprint meeting, once a sprint is completed, is used to review the work from the previous sprint, and update the status of the project (D. Cohen, et al., 2004). For team size, Cohen et al. say the number of people in a SCRUM team should not be over seven, and should include a programmer, a QA and a technical writer (D. Cohen, et al., 2004).For iteration length, Cohen et al. suggest that a typical sprint should run between one to six weeks, and four weeks are recommended (D. Cohen, et al., 2004).

2.4.2.2. The Principles of SCRUM

Schwaber (2004) concludes the major SCRUM principles, which are small team, adaptability, frequent builds, partitioning of work, and constant testing.

Schwaber (2004) suggests that a small team can provide more effective communication among stakeholders. SCRUM values adaptability (Schwaber, 2004). To ensure the success of the project, the SCRUM practitioners should adjust as the requirements change. For SCRUM project, tasks should be defined and divided clearly and logical. SRCUM also place emphasis on continuous evaluation and testing as the artefacts are still developing (Schwaber, 2004).

2.4.2.3. The SCRUM Master runs the SCRUM project

Hole & Moe (2008) state that the SCRUM approach places emphasis on coordination of projects. Hole & Moe (2008) also point out the importance of SCRUM Masters who are responsible for running the project efficiently. The key responsibility of a SCRUM master is to ensure the development to meet all requirements, and resolve emerging issues. If there is any barrier for the development, part of the SCRUM master's scope is to lead the team to resolve it. However, the SCRUM master does not need to manage other stakeholders. The SCRUM project team members need to be spontaneously self-managed (Hole & Moe, 2008).

2.4.3 Distributed Agile and Distributed SCRUM

As the software industry is globalising, it is becoming more common for a software development teams to work and collaborate together, e.g. outsourcing. However, the practices of Distributed Agile are different from the traditional ones.

2.4.3.1. Challenges

Distributed software development faces many challenges. Therrien (2008) summaries these difficulties into four major categories: time zone differences, communication overhead and cultural challenges, the trust factor among stakeholders, and technical challenges.

Time zone differences make many classic agile practices very difficult, e.g. meetings. For communication and cultural challenges, Therrien (2008) points out physical meetings are the most effective way to communicate among stakeholders. Due to distributed locations, the communication has to rely on other less effective ways. Therrien (2008) also argues the

cultural differences cause barriers as well (Therrien, 2008). For the trust factor, no direct vision for artefacts and roadmap affects trust among different development teams. Apart from these non-technical challenges, Therrien (2008) also mentions some technical challenges e.g. the stabilizability and availability of Internet. Lee and Yong (2009) suggest two more challenges: control and practices used.

2.4.3.2. Distributed Agile Practices

Vax and Michaud (2008) outline a number of distributed agile practices to solve the challenges for distributed development: “using near shore resources”, “strict communication plan”, “shared electronic environment”, “asynchronous retrospectives”.

For near shore resources, Vax and Michaud (2008) state the trust is important among stakeholders across different distributed teams. To enhance the trust (at least between customers and the SCRUM Master), the distributed team should use near shore resources, e.g. onsite customers. For strict communication plan, the number of SCRUM meetings need to be minimised to avoid inconvenience caused by differences of time zones (Vax & Michaud, 2008). Other communication should be set up, e.g. VoIP, and feedback should be provided within a half day. For shared work environment, Vax and Michaud (2008) indicate websites/platforms should be setup to manage the progress of the project, which covers a number of project related data, e.g. updates. For dealing with multiple time zones, Vax and Michaud (2008) suggest to separate assignments and responsibilities for teams in different time zones. For example, if teams are located in three different time zones, then every meeting should only involve the developers from two zones at once. For asynchronous retrospectives, Vax and Michaud (2008) point out that reviews should be done locally. Results should be placed to the shared working environment. Therefore, all team members can see them. For more details, please refer to Vax and Michaud’s work (2008).

2.5. Model Driven Engineering (MDE)

2.5.1 MDE Overview

MDE is a software engineering approach to manage the development of information technology artefacts, with emphasis on reusing computer-aided models to solve domain-

specific problem effectively (Schmidt, 2006). MDE has many merits, e.g. productive, robust, and easy to implement MDE tools (Schmidt, 2006).

The concept of MDE was proposed many years ago (Schmidt, 2006). Many tools and approaches have been developed to support MDE applications. For example, Bull et al., (2006) suggest an architecture to assist developers to visualise software by using MDE (2006). Hosking introduces a set of meta-tools for MDE application development (J. Hosking, 2009). Lew et al., (2004) propose a framework to improve business model transformation and development. In addition, MDE techniques have been used in different fields, e.g. ontologies (Gasevic, Djuric, & Devedzic, 2009). Also, MDE requires constant and effective communication between stakeholders, e.g. project owners, programmers, architectures (Schmidt, 2006). MDE based models are capable of representing different views for different stakeholders (Schmidt, 2006), for example, requirements view, design view. The models become agents between systems and systems, or between designers and systems. The evolution and maintenance of IT artefacts are based on the models and the development of the models and IT artefacts can be separated (Schmidt, 2006).

Microsoft introduced a similar idea to MDE: software factories (Greenfield & Short, 2003). Greenfield defines software factories as “the industrialization of software development” (Greenfield & Short, 2003). A software factory has been described as “a software product line that configures extensible development tools like Visual Studio Team System with packaged content like DSLs, patterns, frameworks and guidance, based on recipes for building specific kinds of applications” (Greenfield, 2004).

The MDE and Software Factory concepts are very similar except that MDE usually focuses on supporting open standards. Both approaches advocate working with application domain concepts and introducing automation into the software life cycle. Both emphasize the importance of visual modelling and capturing expertise through patterns. The main difference between the approaches is the emphasis that is put on open standards, particularly UML.

With the evolution of MDE, more people have realized the effectiveness of using MDE in software architecture. Software programmers emphasise the robustness and simplicity of computer aided models at the architecture level, which has become one of the software industry common practices (Schmidt, 2006).

MDE has been developed into a number of research branches. Nowadays, in software engineering area, MDA gains more popularity and usage both in software industry and academic research communities (OMG, 2009).

2.5.2 Model Driven Architecture (MDA) Overview

MDA is a method to design and develop Information and Technology artefacts, e.g. software applications. Model Driven Architecture (MDA) is an open MDE standard proposed by the Object Management Group (OMG) (2009). OMG states that MDA divides business logic, applications from platforms (OMG, 2009). Brown (2004) highlights that there are three major principles for MDA: direct representation, automation and open standards.

MDA solutions focus on particular problem domains and mock up information artefacts using computer aided models. The OMG states that "the MDA is a new way of developing applications and writing specifications" (OMG, 2009). OMG also indicates that MDA projects put emphasis on functionality and behaviours of information technology artefacts that will be deployed in different platforms (2009). Brown (2004) outlines a set of MDA based models, which are a computation-independent model, and a platform-independent model and a platform-specific model. Computation-Independent Model (CIM) models are used to define specific problem domains that for an enterprise environment, which facilitates application design from requirements analysis. Platform-Independent Model (PIM) defines how to build applications without concern to particular platforms. Platform-Specific Model (PSM) models are built for specific platforms. One key philosophy of MDA is how to convert PIM modes to PSM models. Brown (2004) states the PIM models are high level models, and PSM models are low level instance style models. For different models, PSM models provide details and reference of technologies for platforms (A. Brown, 2004).The behaviours and functions of the systems are separated from implementation. New technologies can be easily integrated into PSM models with updating PIM models, which dramatically increase productivity, reusability and portability (2009).

By using MDA, software architectures only need to focus on extracting key issues from the problem domain and represent them with high level models using different modelling techniques (2009). The problem domain models are solved and converted into solution models. There are many advantages of using MDA approaches. For instance, when the

requirements of the problem domain changes, the software architectures can easily track and update decisions. OMG (2009) identifies some major advantages. First of all, by using MDA approach, software developers can focus on the different views and relationships among the major businesses requirements and the enterprise processes, instead of less important issues such as the usage of technologies. The latter are managed by other development platforms and services. Secondly, an MDA driven project normally uses a multi layer design (e.g. 3-tier, N-tier). The specific applications are supported and assisted by middleware and the architecture at the back end. MDA is more convenient for developers to integrate different technologies for different purposes or platform without changing the original architecture design. Finally, OMG points out (2009), the domain concept increases interoperability to MDA.

Brown (2004) points out a few rules for using MDA. Firstly, to extract the essential elements of businesses applications, it is important to develop sound notations which provide a solid representational foundation. Secondly, the models can be used in different orders for different views or different purposes and they can be arranged in different layers. The models are independent from any applications or any architecture. The models provide basic building blocks of well-designed artefacts. Thirdly, the meta-models are models to describe models. The meta-modelling can be used to assist and improve the development and design of the models. Lastly, for MDA, there are many modelling methodologies and standards. It is also important that the development of MDA should stay open and live to the fasting changing requirements (A. Brown, 2004).

To help software developers to understand applications, Brown (2004) suggests UML can be used to visualize the designing software applications before conveying the design models into the real software applications. In recent years, research communities have spent lots of effort on MDA for Domain Specific applications. Furthermore, OMG highlights the benefits of using MDA in domain specific software. For more details about MDA, please refer to OMG's documentation.

2.6. Related Research

2.6.1 Onto-SCM

Ye, Yang, Jiang, & Tong (2008) present an ontology-based supply chain management solution called Onto-SCM. They state the objective of developing Onto-SCM is to provide a vendor independent supply change managing modelling environment with a set of pre-defined Supply Chain theories and terminologies, which are commonly available. They use IDEF5 for visual interface design and Ontolingua to describe the semantics of the models and knowledge communication and interconnection support. They highlight several advantages of this tool. Firstly, Onto-SCM overcomes incoherent terminologies for supply chain partners. Secondly, it improves the effectiveness of knowledge sharing. Figure 2 shows the interface of Onto-SCM.

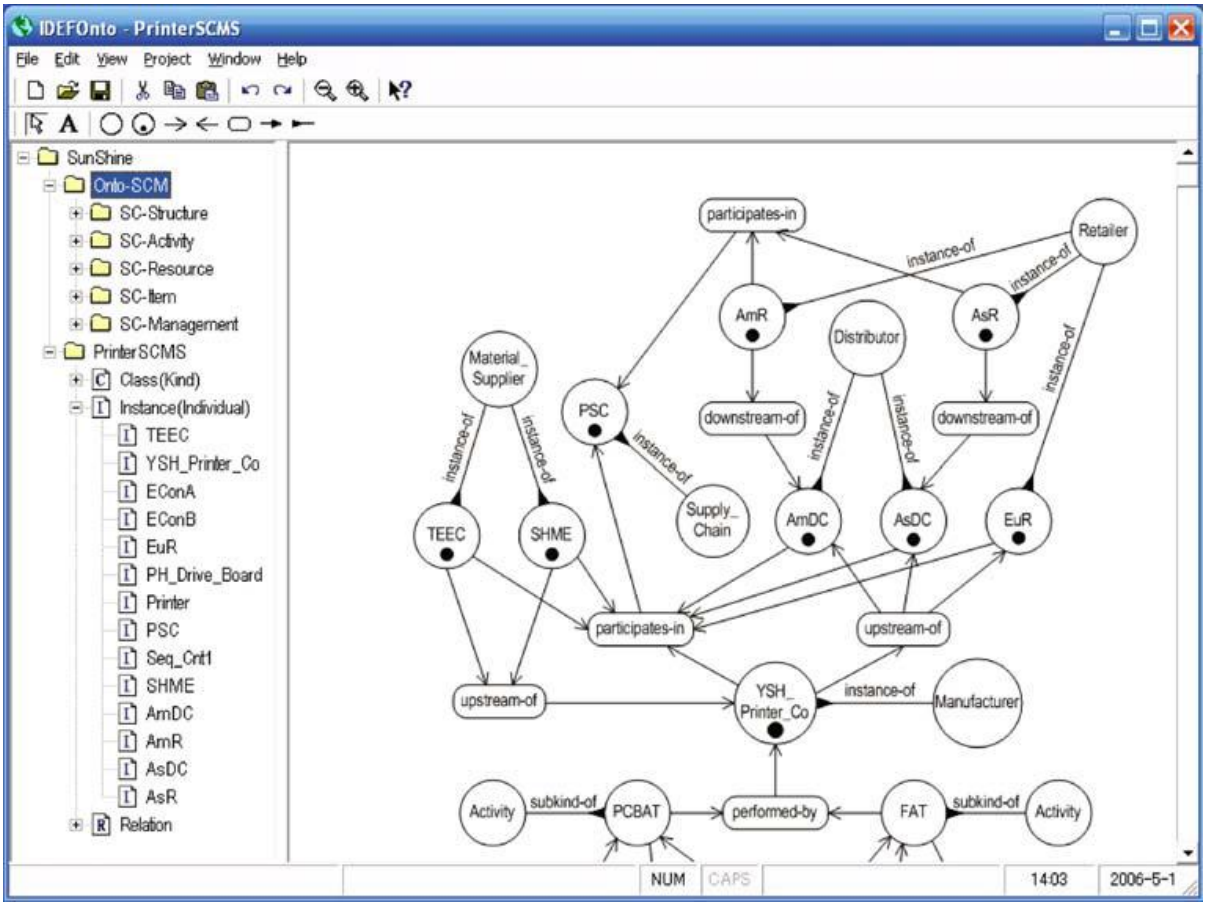

Figure 2. The visual interface of Onto-SCM. From ("Ontology-based semantic models for supply chain management", by Ye, et al., 2008, 2008, The International Journal of Advanced Manufacturing Technology, 37, pp. 1258)

Ye et al. (2008) state that IDEF5 tools are developed by Knowledge Based Systems Inc (KBSI). IDEF5 comprises of two languages: "the IDEF5 schematic language and IDEF5

elaboration language". They say the IDEF5 semantic language is capable of supporting visual notation design and these visual metaphors are used to represent the actual ontology classes. Figure 3 shows a number of visual symbols were designed by using IDEF5, including a number of generic shapes and lines to express different meanings.

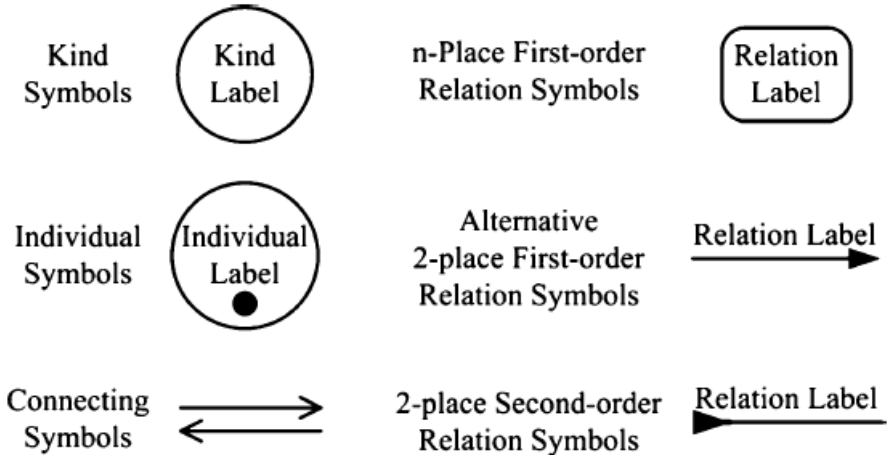

Figure 3. The visual symbols of Onto-SCM. From ("Ontology-based semantic models for supply chain management", by Ye, et al., 2008, 2008, The International Journal of Advanced Manufacturing Technology, 37, pp. 1251.)

Ye et al. (2008) report that the IDEF5 elaboration language is "a structured textural language". Since this elaboration language is developed on top of Knowledge Interchange Format (KIF) and KIF supports first order logic, the elaboration language only provides first order logic semantics. However, IDEF5 ontologies are complex and difficult to learn and it does not support rich features (Ye, et al., 2008).

Ye et al. (2008) assert that Ontoligua is the best ontology language in terms of "the representation of concepts, taxonomies of concepts, n-ary relations, functions, axioms, instances and procedures". Also, Ontoligua is able to communicate with different types of ontologies. They argue that Ontologiua is better than the IDEF5 elaboration language because it supports modularity construct. By using Ontolingua, they were able to define classes, relations and functions. They state that five supply chain theories are modularized by using Ontoligua: structure theory, activity theory, resource theory, item theory and management theory (Ye, et al., 2008).

Ye et al. (2008) argue that supply chain partners normally use different application, which makes it difficult to communicate among partners. Onto-SCM provides a common communication platform to allow different applications to share knowledge by using pre-

defined terminologies. Figure 4 shows different ontologies from different partners can communicate each other semantically by using Onto-SCM .

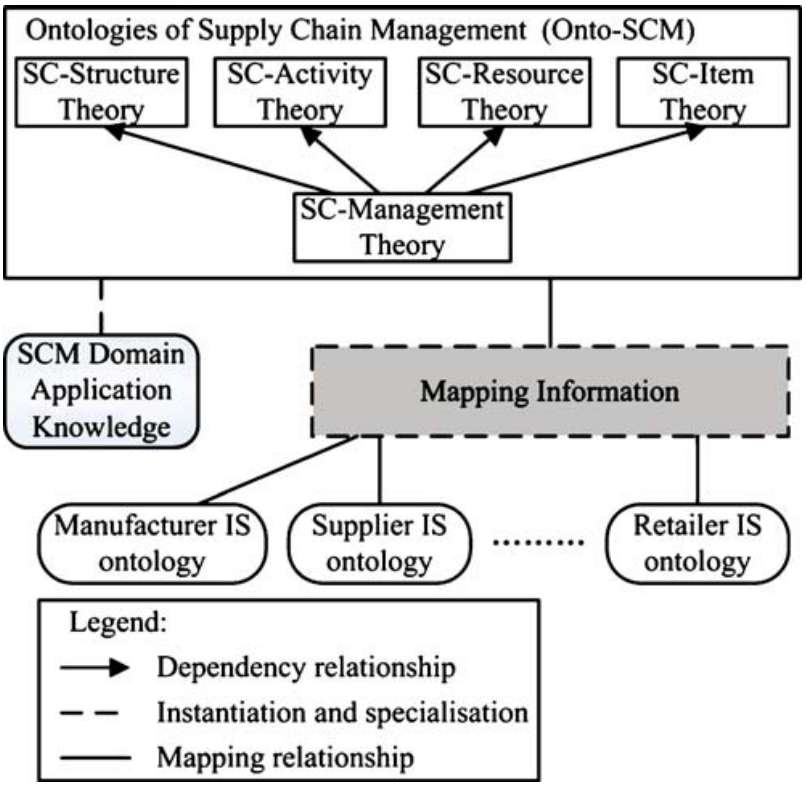

Figure 4. The structure of Onto-SCM. From ("Ontology-based semantic models for supply chain management", by Ye, et al., 2008, 2008, The International Journal of Advanced Manufacturing Technology, 37, pp. 1252)

We felt that Onto-SCM addresses the importance of knowledge sharing and interoperability among supply chain partners. However, it also has some drawbacks. The main drawback of this platform is that IDEF5 has not been popularly used since it was introduced. Also IDEF5 does not support the semantic web, a machine-readable semantic format. The second drawback is the limitation of the graphic representation support in IDEF5. As a result, Onto-SCM only generates some restricted visual metaphors, which we believe are inadequate to perform complex supply chain modelling.

2.6.2 Supply Chain Simulation Tool

Cope, Fayez, Mollaghasemi, & Kaylani (2007) proposed a supply chain simulation solution, which was developed into a prototype tool for NASA's space exploration supply chain

project. The tool is equipped with a GUI interface, Supply Chain simulation ontologies, and an automatic model generator. The objective of this simulation tool is to provide analysis and modelling support.

Cope et al. point out that since the SCORE model is de facto the Supply Chain industry standard model, they used the SCORE model as the foundation of their simulation ontology. Cope et al. state that the SCORE model does not support simulation. Therefore, they inserted "a Resource class, Processing Duration class, Simulation Setup Class and Entity Class". Add-on classes are used to describe different knowledge, e.g. resources.

For the format of their simulation ontologies, Cope et al. only mention that they are "implemented using XML and XML schemas" and that the schema is used to define and customise the supply chain models.

To generate a simulation output, users select options (e.g. structure) via a generic GUI interface. Based on the user's selections, the tool retrieves knowledge from the ontology and populates the output with the retrieved information (e.g. instances). This process is done by an automatic generator. Cope et al. state that the GUI tool is platform independent. It facilitates users to store ontologies and scenarios files. The population process combines the users' selection with a conceptual model to generate scenario-specific simulation models. Figure 5 shows XSLT is used to transform the conceptual model and users' selections into an xml file.

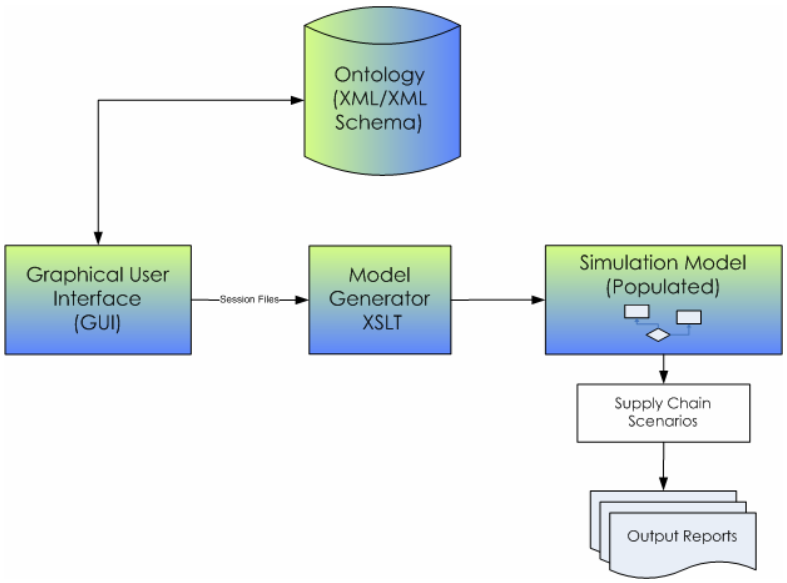

Figure 5. The structure of the simulation tool. From ("Supply chain simulation modeling made easy: an innovative approach", by Cope, et al., 2007, Proceedings of the 39th conference on Winter simulation, Washington D.C, IEEE press.)

Figure 6 shows the generated models are parsed to produce an xml file in a generic GUI interface.

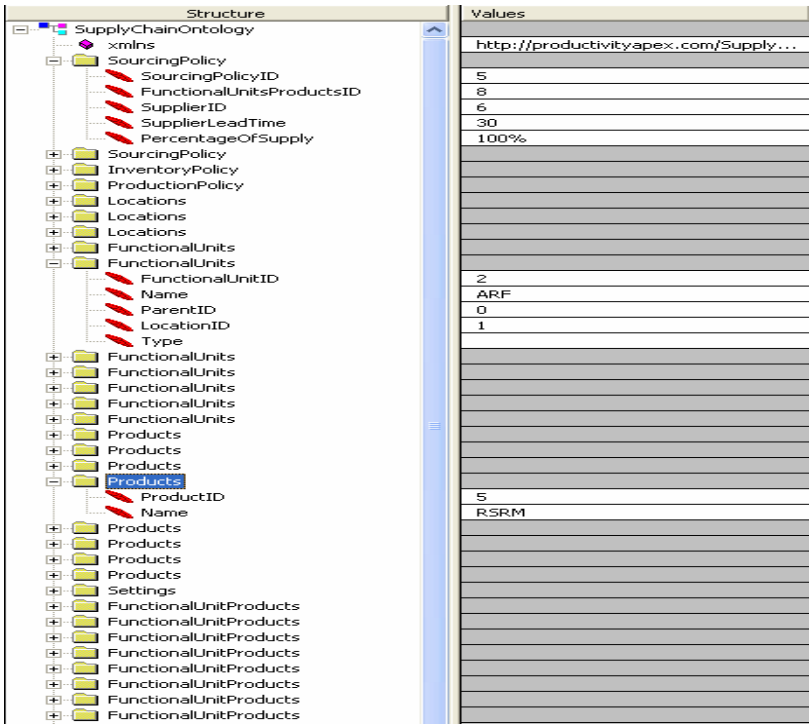

Figure 6. The Generic GUI Interface. From ("Supply chain simulation modelling made easy: an innovative approach", by Cope, et al., 2007, Proceedings of the 39th conference on Winter simulation, Washington D.C, IEEE press.)

Cope et al., state that they use Arena modules to obtain the final models by using VB routines. They did not show any final output in their work. Therefore we do not comment on the results of the generated model.

We felt that the design of the simulation model generation is straightforward, which makes it easy to use and easy to implement. Also the proposed solution addresses the possibility of the simulation model generation by using pre-defined ontologies and scenario based users' input. However, we feel that the major drawback is that the tool uses XML schema to develop the ontologies. Since their custom XML schema does not support semantic web, if the ontology need to be understood by machines, they have to map XML schemas to other semantic web format (e.g. OWL) and the mapping between XML schema and the semantic web can cause inconsistency, which is not perfect for knowledge sharing.

2.6.3 Other related tools

Chatfield, Harrison, & Hayya (2009) propose an open information framework (a Visual Supply Chain Editor), which is based on a XML mark-up language for storing supply chain structure and information. Chatfield et al. argue that the framework provides a common format to improve the quality of the supply chain description. This approach, however, lacks sufficient visual support at the meta model level.

Gabriel Alves, Maciel, & Lima (2007) propose an approach that uses predefined components to support high-level model design. Then the high-level model can be converted into a Generalized Stochastic Petri Nets model. The tool was implemented by using Eclipse GMF. It bridges the GEF and EMF by creating GEF editors that show data from models managed and generated by using Eclipse EMF. It is similar to the Marama meta-tools in terms of its usage of an Eclipse plug-in approach. But it does not support any multi-view or multi-user.

The visualization of complex concept structures and models is a key component of supporting tools for many applications in science and engineering. Diagrams are often used to represent knowledge that can be modelled as objects and connections between those objects, e.g. UML. The visual representation of knowledge is easier to be understood and to be read for human users. Some work has been done for the visualization of tool support. For example, MaramaDPTool (Maplesden, Hosking, & Grundy, 2001) is a tool for visually modelling and instantiating design patterns into UML designs. MaramaDPTool uses a visual language framework to provide support for the incorporation of the solutions proposed by designing patterns into software.

2.7. Discussion

Our literature review came across several different fields (e.g. Supply Chain Modelling, VL design, MDA). Some of these reviewed materials are descriptive, and some of them are analytical. To establish a conceptual infrastructure for our work, we weaved these threads together to facilitate us in different stages, e.g. requirement analysis and visual language design. After reviewing the literature, we learnt a few important lessons from different fields. They helped us to draw an overall picture of other researchers' work and identify our research values. We discuss these lessons we learnt as below, which were also addressed at different stages and aspects of our research.

First of all, we learned how the MDA approach can improve the maintainability and evolvement of software tools by separating domain models from platform dependable applications. We learnt that the MDA approach requires more effort and time on maintaining different models (e.g. PIM, PSM). In other words, developers need to do more work to ensure model consistency over different models over time. However, the advantages of MDA overweight this trade-off regarding in long time evolvement and maintenance. The MDA approach is an important aspect for the architecture of our tool. And the MDA approach established a primary theoretical framework for our requirements analysis, tool design. During our tool design, by using MDA approach, we successfully separated different models into different layers. For the details of using MDA approach in our tool, please refer to our Tool Design Chapter.

Secondly, we learnt both the CDs framework and the Physics of Notations framework are important in visual language research community. We learnt that the strengths and the limitations of both frameworks for visual language design. Also the literature review shows the key characterises of visual languages and how visual languages have been used for human and computer communication and interaction (Moody, 2009a). And we should use "explicit design rationales" to guide our visual language design (Moody, 2009a).

Moreover, we learnt how DSVLs can facilitate computer users in particular problem domains as a special kind of visual language. We learnt that the Physics of Notations framework (Moody, 2009a) is a revolutionary software engineering-based visual language design framework, which comprises of different sets of design principles and theories. These design principles have been addressed in our visual languages design practice. We learned that the CDs framework can help us for our visual language development, especially self evaluation. By studying both design frameworks, we learnt the importance of visual syntax among the visual icons of visual languages which has been ignored by the software engineering research community (Moody, 2009a). Therefore, we emphasised both visual syntax and visual semantics in our visual language design. Both visual language design frameworks facilitated our visual language design. For details of using both the Physics of Notations and the CDs frameworks, please refer to our Visual Language Design Chapter.

Thirdly, we learnt the state of the art of supply chain modelling with a variety of proposed modelling approaches and solutions. We understood the purposes, roles and capabilities of supply chain models (Min & Zhou, 2002). We learnt what role our tool should play in supply

chain modelling to best suit our target users. By obtaining domain knowledge from our literature review, we were able to communicate with the domain experts by using same "domain languages" (domain terminologies), and ensured us to refine requirements that obtained from the domain experts. As we discussed earlier, the current available supply chain modelling tools and approaches only focus on certain issues for supply chain modelling (e.g. decision support). There is no good solution available to facilitate SMEs users to participate supply chain modelling over dynamic supply chain environment. To assist supply chain modelling practitioners in terms of knowledge interoperation, some approaches have been proposed to use semantic web (e.g. XML schema). However, due to the constraints of the knowledge format used in these tools, the knowledge sharing by using these supply chain modelling tools still have compatible and consistent problems (e.g. inconsistency issues).

Furthermore, most of these modelling tools are lack of sufficient visual modelling interface support for complex supply chain models (some of them only provide generic form-based interfaces). Effort is required for scientific-based rapid visual language prototyping to address software engineering based design methodology. Regarding the scope of supply chain environment, our tool needed to facilitate users in a range of key supply chain features, which includes uncertainty, dynamics and distribution (Cope, et al., 2007). By doing this review, we learnt the gaps among these proposed supply chain modelling support solutions, which are also our research focuses. And we identified and outlined the scope of our work: scientific design based visual interfaces assist SMEs users to model complex supply chain models; and end users' knowledge needs to be shared and exchanged effectively and semantically.

Finally, our literature review indicates many distributed projects failed, due to poor project management or poor communication, or lack of trust among distributed development team members (B. Cohen & Thias, 2009; Kontio, Hoglund, Ryden, & Abrahamsson, 2004; Therrien, 2008). Our literature review summarised the challenges that are faced by distributed software development projects. To ensure the success of our tool development project, we did a systematic study about the agile software development methodologies for distributed projects, especially XP practices and distributed SCRUM. We learnt that some work has been done for solving a number of problems for GSD. We understood there is no single silver bullet that can solve all issues. We must use a synthesized approach to manage our project and overcome the problems of our tool project. Our systematic literature review helped us to understand all classic agile practices and approaches (e.g. short release, as simple as possible,

and customer feedback) and provided us a conceptual framework to manage our tool development project.

2.8. Summary

In this chapter, we identified a number of key technologies and techniques which are related to our work. After performing this literature review, we learnt lessons from a number of theoretical frameworks for the architecture of our tool, visual language design and etc. These lessons learnt from tour literature review were very useful for our requirements analysis and tool and visual language design. By exploring the state of the art of supply chain modelling and related support tool and solution, we also identified our research value, which focuses on using visual languages and supporting tool to improve the supply chain management and interpretability of knowledge.

Chapter 3. Requirements and Specifications

3.1. Introduction

3.1.1 Roadmap

Requirements engineering is very important for software development since it can help stakeholders and developers create effective visions of software projects through the whole development life cycle (Paetsch, Eberlein, & Maurer, 2003). To facilitate coordination among Small-Media-Enterprises (SMEs) in supply chain networks, a complex conceptual model has been developed by EC-funded SUDDEN project (Mehandjiev, et al., 2009). SUDDEN stands for SMEs Undertaking Design of Dynamic Ecosystem Networks, which provides a infrastructure for value-added process coordination and supply chain management for SMEs (Mehandjiev, et al., 2009).

To assist end users to use this conceptual model, a domain specific visual language and its support tool (MaramaSUDDEN) has been developed (John Hosking, et al., 2008). In the initial development of MaramaSUDDEN, REEP was adopted to provide a complex view (John Hosking, et al., 2008). However this had shortcomings in the language design and tool support and further evolution of the SUDDEN model made a REEP approach impracticable and necessitated a different approach. As the SUDDEN project has continued development, a number of concepts and relationships were being developed into the SUDDEN model. These challenged the REEP approach taken in the initial design through the need to cater for additional cross-cutting concerns. MaramaSUDDEN needs to be extended to cover recent changes to the domain model. This requires upgrades to both the meta-models and the surface notations of MaramaSUDDEN.

In this chapter, we discuss both functional and non-functional requirements, and capture user requirements. The main requirement of the current development is to cover the whole theoretical model and to increase the usability of the modelling tool. Moreover, both the MaramaSUDDEN visual language and support tool should be maintainable and extensible for long term evolvement. We used different requirement engineering techniques to elaborate the requirements for our tool, e.g. interviewing, use case modelling (Arlow & Neustadt, 2005).

In addition, we organize this chapter into different sections. We firstly present an introduction which includes a roadmap, aims of this chapter, our approaches, and how we document the identified requirements. Secondly, we discuss the domain model, and identify requirements for our visual language design in terms of the meta-models and visual notations by exploring different subsystems of the domain model. Thirdly, we present the requirements for the visual modelling environment, which includes the functional requirements and non-functional requirements. Finally, we summarise this chapter's work. Figure 7 shows the roadmap of chapter 3.

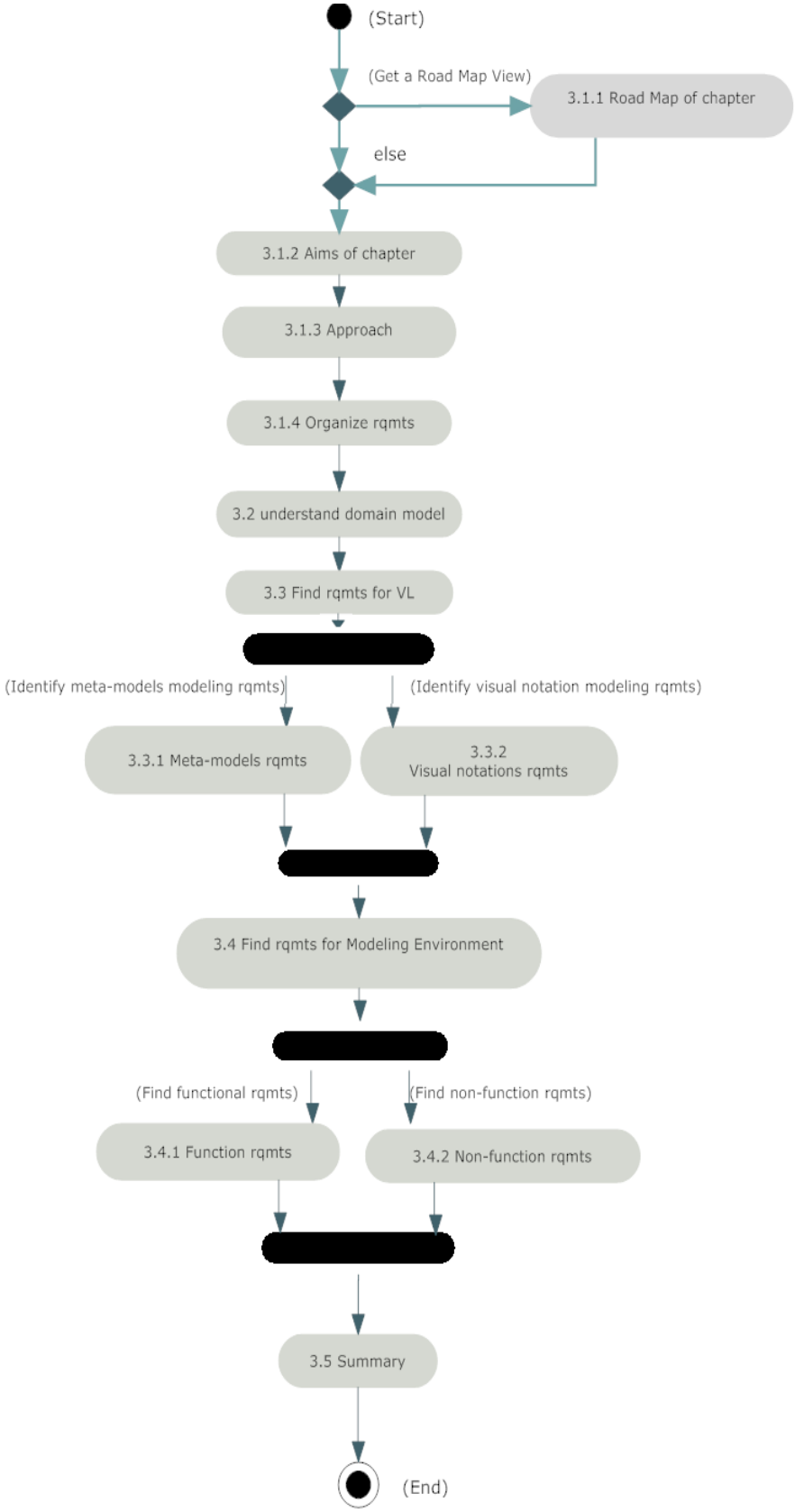

Figure 7. The Roadmap of Chapter 3.

3.1.2 Goals and Objectives

Our ultimate goal in this research is to help SMEs to participating in supply chain networks modelling, and share their knowledge. To achieve these goals, there are three key objectives: 1) Report the requirements engineering process for our work. 2) Elicit and identify the users' concerns and design requirements for the MaramaSUDDEN visual language. 3) Identify essential requirements for the MaramaSUDDEN modelling tool in terms of functional and non-functional requirements. By doing this, we were able to establish our sub-goals for the analysis, design, implementation and evaluation accordingly. Since the identified requirements refer to cross-cutting concerns, we grouped the requirements into various aspects. Cuesta, del Pilar Romay, de la Fuente, & Barrio-Solórzano (2005) state that "an aspect is just a kind of module which is conceived to encapsulate the behaviour related only to an specific concern".

3.1.3 Approach

A synthetic approach was used for our project development, which includes the Unified Process (UP), distributed SCRUM and XP. UP is an incremental iterative process (Arlow & Neustadt, 2005). We adopted UP as a primary conceptual framework to aid us for analysis, design and evaluation. Arlow & Neustadt (2005) summarize that the key phases of UP, which includes Inception, Elaboration, Inception and Inception. Our requirements analysis was mostly done in Inception and Elaboration phases. To gather requirements, different requirement engineering techniques have been used, e.g. interviewing, use case modelling. We used distributed SCRUM to manage and organize our project. We used the XP methodology (e.g. short release) to improve our productivity for the development. Further details on using agile methodology in our work, please refer to our software engineering 2.0 chapter.

Our requirement discovery was a goal-driven process (van Lamsweerde, 2001). We decomposed the primary goal into different aspects. For the each of the aspects, we identified and classified the corresponding requirements for it. To identify and determine requirements, some of the corresponding questions have been developed from different point of views. The requirements of each aspect were explored through a synthetic process of answering these

questions, and interviews and discussion with the stakeholders. Some examples of the questions for our requirement analysis are as follows.

- Who is going to use our tool?
- Who is the target user?
- Who should we engage?
- Which user needs will the product address?
- Which product attributes are critical to address the user needs?
- What is the domain model?

Initially, we considered using some requirements engineering tools. However this could generate additional work for stakeholders to learn how to use them. Hence, we decided not to use any of those tools for the current development.

3.1.4 Documenting Requirements

We identified the major requirements at the beginning of our project. The detailed requirements were refined in Inception and Elaboration phases through each of iterative development life cycles (UDLC). The identified requirements are the essential input for our DSVL and tool design, implementation and evaluation processes. Hence, it is critical to document and organize these requirements properly for future development and maintenance. Figure 8 shows the syntax and mechanism for our requirements engineering modelling.

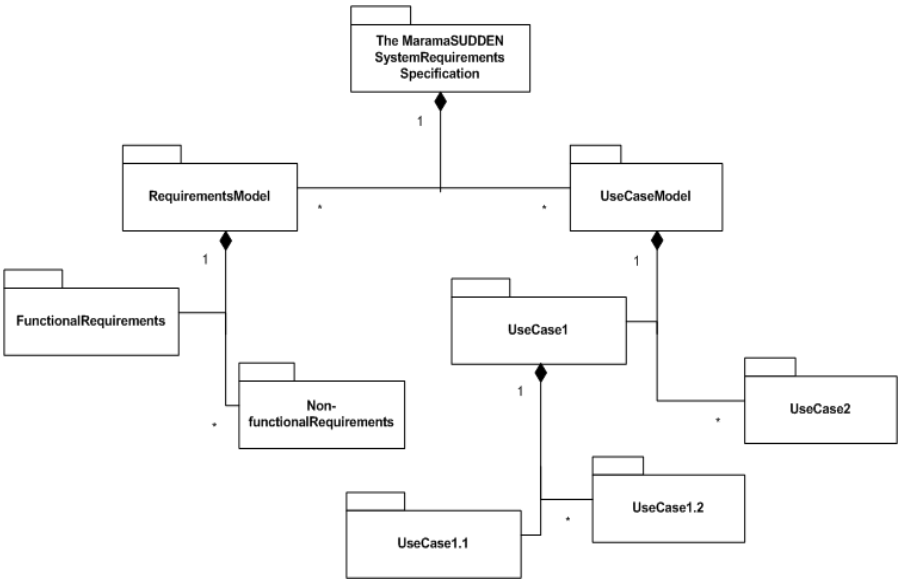

Figure 8: The syntax for our requirements engineering models. Adapted from ("UML 2.0 and the Unified Process: Practical Object-Oriented Analysis and Design " (P.52), by Arlow & Neustadt, 2005, 2005, Addison-Wesley Professional.)

To document these identified requirements, we used a set of requirement attributes that indicate the priorities for our development work in the given time frame and resources. We used MosCoW scheme (Arlow & Neustadt, 2005) to mark all identified requirements. And these attributes carry different values, which indicate the importance for development against the current development period. The MosCoW priority attributes include "MustHave", "ShouldHave", "CouldHave" and "WantToHave" (Arlow & Neustadt, 2005). "MustHave" and "ShouldHave" specify the top priorities, which tasks need to be done in the first order. "CouldHave" and "WantToHave" are not urgent, which can to be implemented in the future releases (Arlow & Neustadt, 2005). Table 1 shows the priority attributes for our requirement documents.

Attribute Values for the MaramaSUDDEN Requirements	
1. MustHave	Top Important(critical)
2. ShouldHave	Second Important (crucial)
3. CouldHave	Less important
4. WantToHave	Future Release

Table 1. The Priority Attribute values for the documents of MaramaSUDDEN requirements. Adapted from ("UML 2.0 and the Unified Process: Practical Object-Oriented Analysis and Design " (P.59), by Arlow & Neustadt, 2005, 2005, Addison-Wesley Professional.)

For software engineering, there is no single industry standard requirements model (Arlow & Neustadt, 2005). In our case, we chose to describe the identified requirements in text. However, we used simple formatting approaches to keep the requirements document maintainable. If we choose to employ a requirements engineering tool in the future, we believe the requirements document can be parsed properly. The format of a requirement

model document is <id> <the module> shall<function>. The <id> is a unique identifier, "shall" is the keyword, and <function> is the function to be performed. If it is a non-functional requirement, <non-function> is used in the requirement model document. According to Arlow & Neustadt (2005), non-functional requirements indicate "constraints" for software tools. Arlow & Neustadt (2005) give some examples for these constraints. For example, "The ATM system shall communicate with the bank by using 256-bit encryption" (Arlow & Neustadt, 2005).

3.2. Understand the Domain Model

To design successful visual notations and modelling tool, we need to understand the domain model, which is one of essential source for requirement gathering. With stakeholders' active involvement, we organized a number of interviews and workshops with them. And requirements were elicited from discussions, questions-and-answers interaction.

Mehandjiev, et al. (2009) describe the SUDDEN model as "bringing together ideas of collaborative planning and design of abstract supply networks, delayed partner recruitment and systematic evolution of supply networks". The domain model addresses value creation among collaboration and coordination of SMEs participated supply chain modelling. Also, the domain model provides a conceptual platform to support SMEs to share, learn and create domain-specific knowledge. Figure 9 illustrates the key concepts and mechanisms of the domain model.

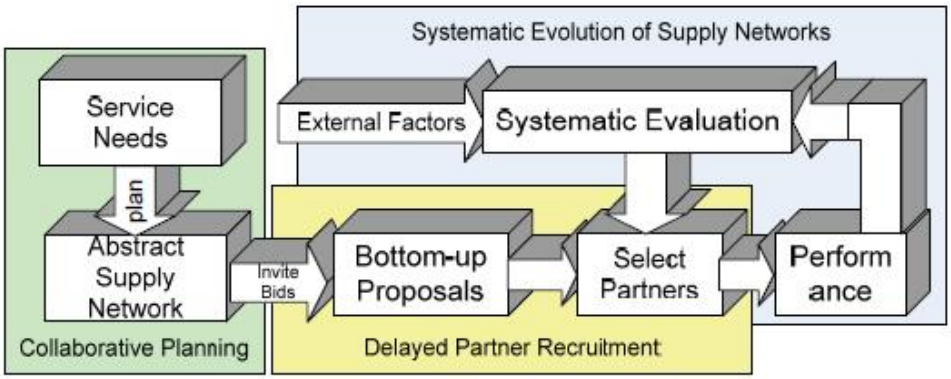

Figure 9: The key concepts and mechanisms of the domain model. From ("Recursive Construction and Evolution of Collaborative Business Processes" Mehandjiev, et al, 2009, p.3.)

In Figure 9, the domain model comprises of a number of subsystems. Mehandjiev, et al. (2009) highlight that "the collaborative planning subsystem conceptualises supply networks as comprising actors which provide services (including the provision of goods), and then decouples service requirements from the concrete actors chosen to fulfil them". It reflects some of key principles of the domain model, which facilitates SMEs to find new business opportunities from forming dynamic business collaboration within visual enterprises (VE) (Mehandjiev, et al., 2009). The delayed partner recruitment mechanism addresses dynamics of selecting and changing supply chain partners, and forming a supply chain team. Mehandjiev, et al. (2009) state "Bottom-up Proposals allows suppliers to form consortia which offer innovative combinations and bundling of activities". The "Partner Selection" is also dynamically changing based on "performance" and "external factors". In addition, both "external factors" and "performance" are input of systematic evaluation of "Partner selection". Mehandjiev, et al. (2009) state the domain model "focuses on the knowledge-based model and model construction operators which can provide the general framework integrating the mechanisms above" (Mehandjiev, et al., 2009).

Based on our domain model study, we felt the concepts of the domain model are complex and rather abstract to represent. The domain model embodies a large number of cross-cutting concerns, and collaboration among different subsystems. After interviewing the stakeholders, we understood there are three key sub-systems of the domain model, which are a Process subsystem, a Product subsystem and an Abstract Supply Network (ASN) subsystem. In addition, the domain model has many relationships between different concepts. The relationships carry different semantic meanings. After discussion with the stakeholders, they requested that they need three separate interfaces for each of these subsystems, which is our first requirement as below.

- <3.2> [1]the visual notations and their modelling environment *should* have three different visual interfaces for handling each of these subsystems, which includes the Product subsystem, the Process subsystem, and the Abstract Supply Network subsystem.

[1] The IDs of the identified requirements are numbered by the sequence of the sections and subsection of this chapter, which is easy to follow.

3.2.1 The Product subsystem

The Product subsystem focuses on decoupling product for users to explore potential business opportunities to create new value. The key activities of this subsystem are to facilitate SMEs for creative thinking (Mehandjiev, et al., 2009). The product subsystem provides a conceptual context for SMEs to think about different ranges of potential decompositions of product in order to produce new value. As a result of using this approach, SMEs can explore and find new businesses at lower price within their capacities (Mehandjiev, et al., 2009). For example, a supplier company currently supplies electronic motors for electronic car door window systems. It is possible for this company to join a supply chain of electronic mirrors to supply similar electronic motors within a virtue enterprise system. Figure 10 shows the conceptual model for the Product sub-system of the domain model.

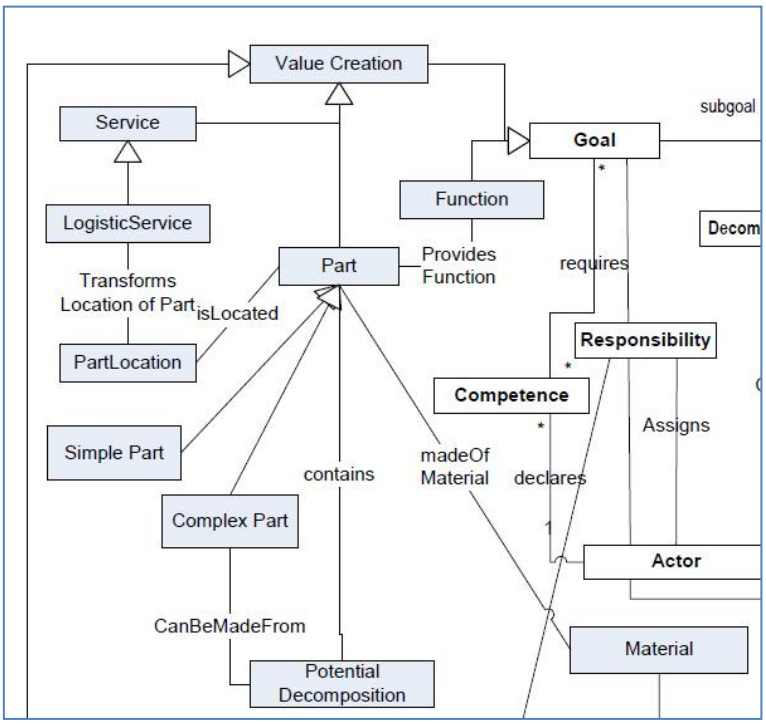

Figure 10. The conceptual model for the Product sub-system. (This drawing is credited to Martin Carpenter, from the SUDDEN project, 2009.)

The key concept of the Product subsystem is to treat a specific "Goal" as a value creation process. The value creation process could be via providing services or manufacturing products (Mehandjiev, et al., 2009). And products can provide different functions to create

new values. This is a process to create new value or explore business opportunities (Mehandjiev, et al., 2009).

Another key concept of the Product subsystem is that a product (or a part, the two terms mean the same concept in the domain model) could be potentially made from different combinations of simple parts and complex parts (Mehandjiev, et al., 2009). Hence, a product can be decoupled in different ways. For example, a car door body can be made of different materials, e.g. alloy, a steel. And a car door can be made from different combination of simple parts and complex parts. Different parts of a car door can be further decoupled into a multi-level decomposition. After discussion with the stakeholders, they required that the following domain concepts should be included by the Product interface.

- <3.2.1> the Product subsystem interface/view *should* include representations of the following concepts from the SUDDEN model: Product, Complex Part, Simple Part, Potential Decomposition, Location, Resource, Service, Value Creation, Function and Material.

3.2.2 The Process subsystem

The Process subsystem provides a conceptual supply chain environment for SMEs to plan, coordinate and design their business processes collaboratively within their supply chain networks. And business processes within the dynamic supply chain networks are driven by goals (Mehandjiev, et al., 2009). The Coordination Theory (Malone & Crowston, 2003) is one of fundamental concepts of the domain model (Mehandjiev, et al., 2009) . Mehandjiev, et al. (2009) state that "cross-organisational business processes can benefit from the models and approach of the Coordination Theory".

For example, a "Process" is to make (CarDoors), which depends on another "Process" to make (CarLocks). And the coordination mechanism between the "Process" of making (CarDoors) and the "Process" of making (CarLocks) is "Just-in-time" (a coordination mechanism). When the production of the making (CarDoors) "Process" are increased dramatically, the coordination mechanism between the making CarDoors "Process" and the making (CarLocks) "Process" changes to "Bulk-Order" (another coordination mechanism).

The dependency management of the domain model were adopted from the MIT Process Handbook (Malone, Crowston, & Herman, 2003) There are three generic dependency types of the coordination mechanisms, which are a "Fit", a "Flow" and a "Sharing". Figure 11 shows the dependencies between processes of the domain model. These three generic types are capable to describe the core concepts of the coordination of business processes (Malone, et al., 2003).

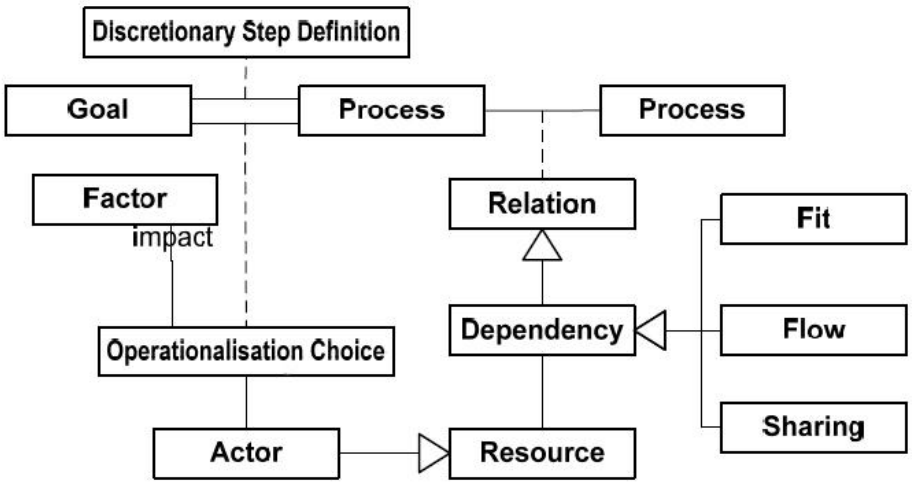

Figure 11. Dependencies between processes. From ("Coordination Models and User Interaction Language" by Carpenter,et al., The SUDDEN project Deliverable D2.1, 2007, p.16.)

Another key concept of this subsystem is "interplay of goal and process" via "the concept of Actor" (Mehandjiev, et al., 2009). Figure 12 illustrates the relationships of Actor, Goal and Process, which are key concepts of this subsystem. As you can see from Figure 12, Goal is the centre of gravity of these conceptual business activities. In other words, in this subsystem, all activities are goal-driven.

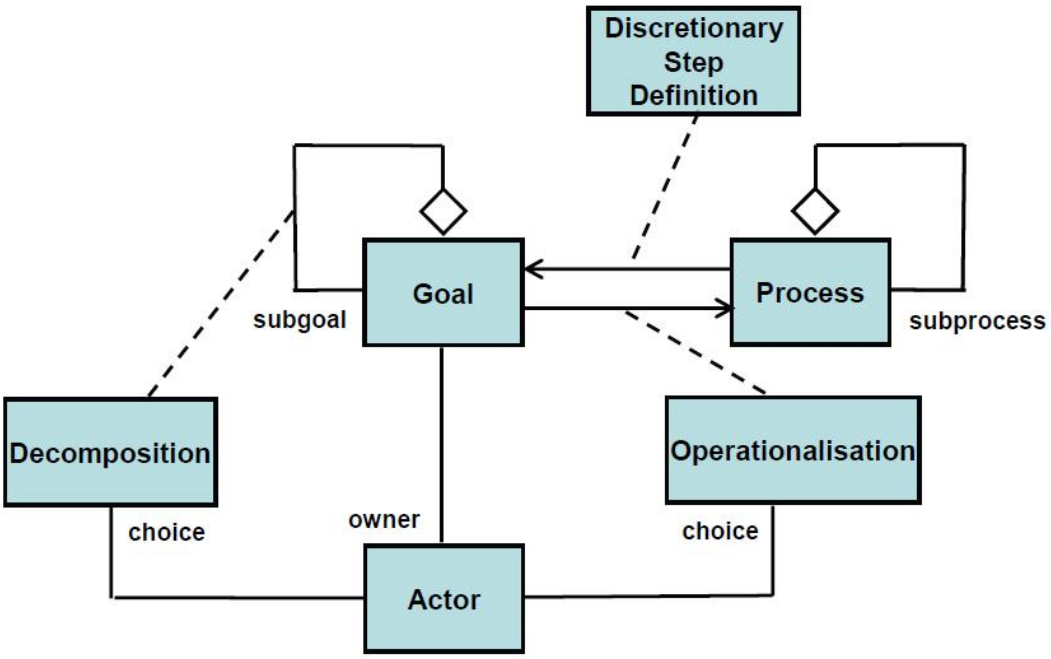

Figure 12. Relationships of Actor, Goal and Process of the domain model. From ("Recursive Construction and Evolution of Collaborative Business Processes" Mehandjiev, et al, 2009, p.6.)

In Figure 12, an Actor can be a company or a person, who is responsible to make decisions for a Goal . For example, a Goal is to make (CarDoors), the decision-maker (a Actor) of this Goal can decouple this goal into a range of sub-goals, which include a goal for making (CarLocks), a goal for making (CarBodies), etc. Or the Actor can make "Operationalisation Choices", e.g., outsourcing or achieving the Goal by certain processes. In addition, to obtain the responsibilities to make decisions for a certain Goal, the Actor needs have certain competencies. And these pre-defined competencies are required by the Goal. Based on interviews and discussions with the stakeholders, the following requirement was identified.

- <3.2.2 > the Process interface should include the following domain concepts: Goal, Coordination Goal, Operationalisation Choice, Competence, Competences Required by Goal, Responsibility, Decomposition, Actor, Machines, Resources, Process, Coordination Process, Processing, Relation, Dependency, Fit, Share, Flow, Factors, factor, and Material Dependency.

3.2.3 The Abstract Supply Network (ASN) subsystem

This subsystem of the domain model is to allow SMEs form an conceptual abstract supply network, which addresses dynamic partner selection and switch by using the concept of visual organization (Mowshowitz, 1997). Within the supply chain networks, each of companies/organizations has been treated as a node of the VE. An ASN can be formed by using an up-to-down approach, or a bottom-up top. The domain model emphasize the bottom-up team formation proposal and late partner recruitment mechanism (Mehandjiev, et al., 2009).

Víctor Blázquez, Martin Carpenter, & Weichhart (2008) state that "bottom up team formation is used when the abstract supply network cannot be initially defined in sufficient detail for top down team formation to operate". In an ASN, an actor can recruit other SME suppliers to form a candidate team. The different teams can be dynamically formed by different combination of suppliers. And each of suppliers in the VE can have multiple roles in different supply teams. Each of roles could have multiple responsibilities in different supply chains. The formalisation of ASN also depends of "Resources", e.g. material. In addition, different suppliers within an ASN team also share responsibilities. Figure 13 shows an instance of Abstract Supply Network using the domain model. In Figure 13, a few businesses forms two supply chain teams dynamically.

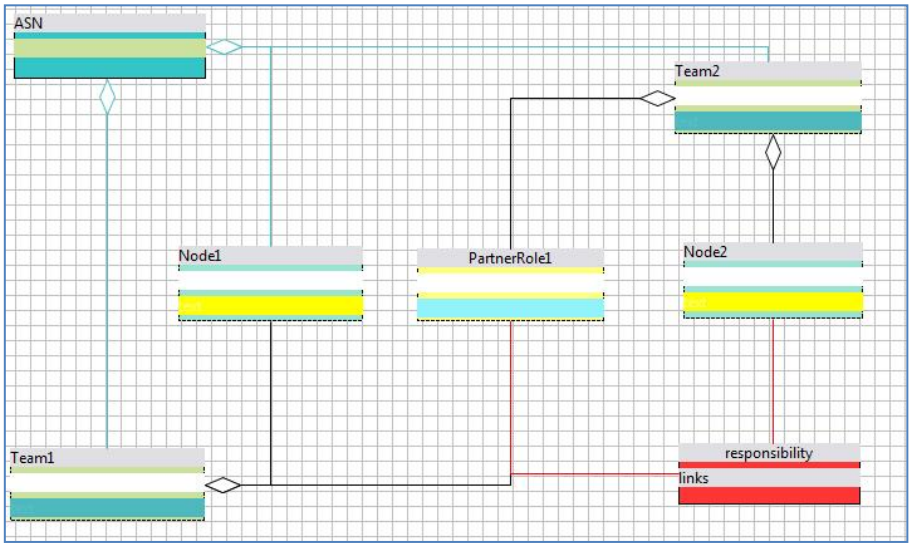

Figure 13. An example of ASN formation.

In Figure 13, a company is a supplier (Node 1) which makes car locks; the second company (a Partner Node 1) is another supplier which makes steel car door body. They form a partial

supply team 1 to potentially supply car doors. This information is visual to all nodes within the supply chain networks. Therefore, the third company (a car lock maker-Node 2) makes an offer to join this supply team. Once the third company joined the chain, a complete supply team is complete which is capable to supply car doors for upcoming business opportunities.

After interviewing and discussing with the stakeholders, the following requirement was identified.

- <3.2.3 >the ASN subsystem *should* include following domain concepts from the SUDDEN model, ASN, Team, Node, Partner Node, Dependency and Responsibility. Also the ASN editor also should provide metaphors for the relationships between these concepts.

3.3. The MaramaSUDDEN Visual Language Requirements

In this section, we discuss the requirements for our visual notation design. We separate the requirements into those which are meta-model modelling requirements and those which are visual notation modelling requirements.

3.3.1 The Meta-Model Modelling Requirements

Atkinson (1997) states that "meta-models enable objects interested in the meaning of the entities they are dealing with to work from the same underlying semantic framework". Venkatesh, Bhaduri, & Joseph (2001) suggest meta-modelling is a promising approach to "describe complex systems in terms of different views". To support our visual notation design, we used meta-models to describe the concept model in higher level, which is independent from all visual symbols and applications. And the visual symbols and the meta-models are in different level, the evolvement of both models can be totally separate. Below are some requirements for the meta-modelling of our visual languages.

- <3.3.1.1> the meta-models *must* cover the key subsystems of the domain model.
- <3.3.1.2> the meta-models *must* be consistent to the domain model.
- <3.3.1.3> the meta-models *should* describe all domain concepts and relationships as a whole. Therefore, the surface notations can benefit from keeping all SUDDEN concepts together at the meta-model level to keep the notations and interfaces consistent.

- <3.3.1.4> the meta-models *must* describe the essential and basic attributes of domain concepts. For example, all visual notations need an attribute which holds its name; therefore this name attribute is essential for the meta-model.
- <3.3.1.5> each of the domain concepts and relationships *should* have a single corresponding meta-model and their own attributes.
- <3.3.1.6> each of the meta-model *should* have a single- focus responsibility, which can avoid coupling and confusion for future development.
- <3.3.1.7> low coupling is one of desirable features for our meta-modelling, a specific entity class *should* only link to another entity class when it must and avoid unnecessary relationships. By doing this, we would not create any unnecessary dependencies at the meta-model level.
- <3.3.1.8> the meta-models *should* avoid hidden dependencies, which cause difficulties for understanding and maintenance (Zhifeng, 2001).
- <3.3.1.9> each of the meta-models *should* offer a simple primitive responsibility. In another words, the meta-models should avoid complex and multiple operations.
- <3.3.1.10> the domain model has a number of N-nary relationships. The meta-models *should* properly describe these N-nary relationships, which require "the introduction of additional aggregation structures to properly represent them which then often become clumsy" (John Hosking, et al., 2008).

3.3.2 The Visual Notation Modelling Requirements

Due to the complexity and abstraction of the domain model, the UML notations are either not efficient or sufficient for end users to use this domain model (John Hosking, et al., 2008). Moody says "visual notations form an integral part of the language of software engineering". However, visual notation design has been undervalued for many years. (Moody, 2009a). Moretti & Lyons (2005) suggest that software developers are likely to choose the default colours in their visual notation design. To visualise the domain model properly and elicit requirements from the stakeholders for our visual notation design, we organised interviews with stakeholders and reviewed literature on software engineering based visual notation design.

To visualise the domain model, Representational Epistemology (REEP) (Barone & Cheng, 2004) approach was used in our initial design to empower a complex view (John Hosking, et al., 2008). However, as the domain model was still developing, more concepts were being developed. As a result, there were several dozens of concepts and relationships need to be visualised. And one single complex view was not sufficient to provide appropriate modelling space for users. After interviews and discussion with the stakeholders, we decided to split the single complex view into three sub-views to address different aspects of the domain model. Based to the requirements gathering from the stakeholders and the study on Physics of Notations (Moody, 2009b), we documented the following requirements for our visual notations design.

- <3.3.2.1> the visual notation and metaphor *should* use conversion and constraints to provide a feeling of consistency for users (John Hosking, et al., 2008).
- <3.3.2.2> the visual notation and metaphor design *should* use pre-defined visual notation design paradigm and rationales to describe the visual elements (Moody, 2009a).
- <3.3.2.3> the visual notation *could* use modular structure to increase the stability of the screen (Moody, 2009a).
- <3.3.2.4> the visual notation design *should* use strong variations (e.g. different colours and direction arrows) to provide clear distinction (Moody, 2009a).
- <3.3.2.5> the link-type visual notations *should* use well-designed paths to help users to organize modelling space, for example, direct path, shortest path, rectilinear path where necessary. By doing this, the visual modelling space can be optimized and the number of ling crossing can be reduced.
- <3.3.2.6> each concept of the domain model *should* have a primary notation and a secondary notation.
- <3.3.2.7> one visual icon *should* only carry one single meaning to avoid any possible confusion.
- <3.3.2.8> the colours used for our notation design *should* reflect the nature of the concept.
- <3.3.2.9> all visual notations *should* be consistent, which increase the families to our users. For example, colours used for the related concepts should be consistent.
- <3.3.2.10> the link type visual icons *should* use orientations to distinguish to one another (Moody, 2009a). For instance, between "Complex Part" and "Potential Decomposition" there is more than one relationship. One relationship is that a Complex Part can be made

from multiple Potential Decompositions. Another relationship is that Potential Decomposition can contain many different Complex Parts and Simple Parts. Hence, the connections should carry the domain specific meanings "Can Be Made From" and "Contains". Two directions of the connection between these two concepts (Potential Decomposition and Complex Part) should carry different meanings. From a Complex Part to a Potential Decomposition should carry the meaning of "Can Be Made From". From a Potential Decomposition to a Complex Part, the connection should carry the meaning of "Contains". This is extremely important for complex and complicated modelling since the users are likely to be very easily confused.

3.4. The MaramaSUDDEN Visual Modelling Environment Requirements

In this section, we discuss the requirements and the specifications for our modelling environment. We present the tool requirements in two sections: functional requirements and non-functional requirements.

3.4.1 Functional Requirements

Use case modelling is another essential requirements engineering technique to identify and document requirements (Arlow & Neustadt, 2005). We used use cases to understand and document the functional requirements for our tool. In this section, we present how we find the use cases which capture the requirements. We also discuss where the boundaries of the visual modelling environment are, the scope of our tool, and the actors who interact with the visual modelling environment (Arlow & Neustadt, 2005).

3.4.1.1. The boundaries of our visual modelling environment

Before we started to build up the modelling environment, we needed to decide where the modelling environment boundaries lie and the scope of the modelling environment (Arlow & Neustadt, 2005). We believe the present scope of the visual modelling environment is to facilitate users to use the visual notations to collaboratively design and coordinate their business activities. And users' knowledge (e.g. experience, new idea) is saved into the centralised knowledge management repositories for sharing with other SMEs. Our tool also

provides a standardised environment (a suite of visual languages) for SMEs to communicate and interoperate.

3.4.1.2. Actors

To start identifying use cases, we also need to identify the actors to the modelling environment. The direct users are members of the SUDDDEN team. They interact with the modelling environment directly (John Hosking, et al., 2008). The domain experts are also target users for our tool. Both of these stakeholders have common behaviour and use of the modelling environment. We consider them as the same actor group. This actor group has something in common. They are familiar with business process models. They use domain specific jargon. The visual modelling environment can potentially interact with other applications, which can be actors. However, these external applications do not assist us to find the use cases (Arlow & Neustadt, 2005). Therefore, they will be excluded for further discussion.

3.4.1.3. Use Case Modelling

The best way of identifying use cases is to start with scenarios of how the actor is going to use the visual modelling environment (Arlow & Neustadt, 2005). After interviews and discussion with the stakeholders, we created a list of candidate use cases. Then we iterated and refined the candidate use cases through the whole development process. We started with a simple name for each use case. Then we enhanced the use cases with additional details at each iteration cycle (Arlow & Neustadt, 2005). Eventually the details were refined into complete and stable specifications (Arlow & Neustadt, 2005).

Below is a set of stable and complete use cases we obtained, which captured major functions requirements for the visual modelling environment. Figure 14 shows a use case diagram for our visual modelling environment. Table 2 shows a set of use case descriptions and their pre and post conditions.

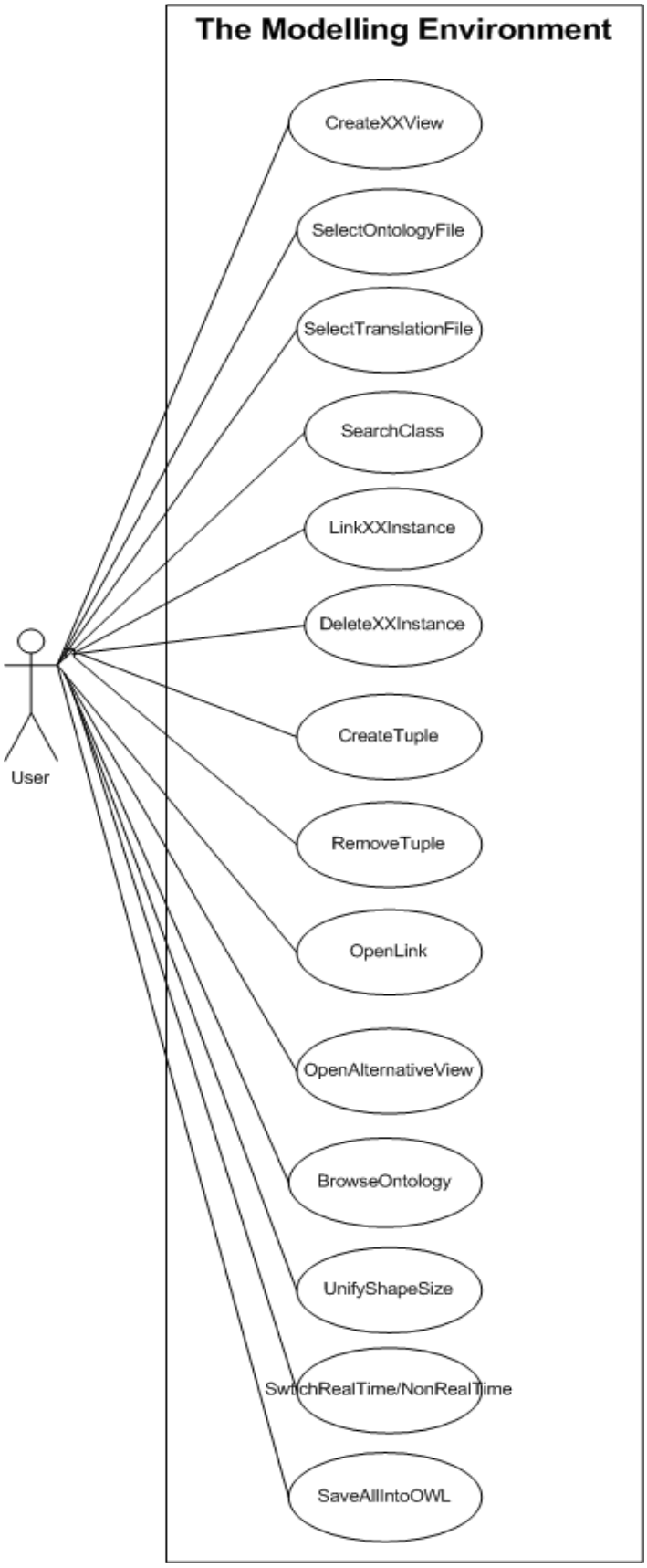

Figure 14: The use case diagram for our visual modelling environment.

Names of Use Case	Description	Pre-condition	Post-condition
• Select & Set a Database	To select a valid data repository, and set up a connection with the visual modelling interface.	None	A valid data repository has been selected and set successfully.
• Select & Set a Translation File	The User selects and set a valid translation file.	None	A valid translation file has been set successfully.
• Search Class	The User searches information for a given class name in the database	A valid connection to a valid data repository.	Return information to users if there are available records in the database.
• Link Instance	1. To link the visual icon to an existing instance in the data, 2. or to create a new instance for a existing class/subclass, 3. or to create a new instance and create a new class or subclass.	A valid connection to a valid data repository. & The modelling environment is in non-real-time editing mode.	1. The visual icon is successfully linked to an instance in the data; 2 a new instance is created under the existing class/subclass, and linked with the visual icon successfully; 3. a new instance and a new class are created in the database, and the new instance is linked to the visual icon.
• Delete Instance	The User deletes an instance from the database.	Same as No.4	The system has removed the instance from the database, and removed the visual icon from the visual modelling environment.
• Create Tuple	The User creates a tuple between two valid instances in the database.	Same as No.4	The system successfully created a tuple in the database.
• Remove Tuple	The User removes a tuple from the visual modelling interface and the database.	Same as No.4	The system has removed the tuple from the database and removes the visual representation of this tuple from the visual modelling environment.
• Open Link	The User opens a hyperlink in a Web browser.	A pre-defined hyperlink is pre-defined.	None.
• Open View	The User opens a modularized object in a separate view.	A valid path to an alternative view is pre- defined.	A different view/diagram has been opened.
• Browse Database	The User browses information from the database.	Same as No.3	None
• Unify Shape Size	To set all visual notation to the default size.	None	The size of the visual icons in the modelling environment has set to the default .
• Switch Real-Time/Non-Real-Time	The User switches the editing modes of the visual modelling environment between the real-time and non-real-time modes. In real-time mode, the data communication with the database is synchronised, in non-real-time mode, it is not.	None	The editing mode of the visual environment has been set based on users' selection.
• Save All	The User saves all his/her work.	Same as No.3	The system has saved.

	into the database.		user's work into the database.

Table 2. A Set of Use Cases for the MaramaSUDDEN Modelling Tool.

Apart from interviewing the stakeholders, we found that use case modelling is a very helpful requirements engineering technique to assist us to find out and document the functional requirements for our tool. Based on these identified use cases, we found out the clear boundaries of our tool, identified major and alternative flows of our too1. In addition, based on these use case modelling, we were able to design the functions of our tool. Furthermore, classes of the essential functions can be modelled accordingly. Moreover, we were able to understand how the actor interacts with our tool. For more details of the detailed specifications of these use cases, please refer to Appendix A.

3.4.2 Non-functional requirements

General speaking, a non-functional requirement is a constraint placed on the visual modelling environment (Arlow & Neustadt, 2005). Arlow & Neustadt (2005) outline a range of non-functional requirements, for example, performance, capacity, availability, compliance to standards and security. In this section, we discuss non-functional constraints for the visual modelling environment in two aspects: GUI requirements and usability requirements. There could some overlapping between requirements between both aspects. However, we need to consider these requirements from different points of views. Therefore, we reckon some overlapping between the two aspects is acceptable and healthy.

3.4.2.1. GUI design requirements

Since our tool is a visual modelling environment, user interaction is one of our major concerns. After interviewing our stakeholders, they requested to use extra GUI windows to obtain more data from the repository for user control and interaction. Below are some user requirements for our extra GUI window and message design.

- <3.4.2.1.1> the visual modelling environment should keep all GUI design consistent. For example, our modelling environment should allow our users to learn something from one function, then apply the same knowledge for other functions in the same environment.
- <3.4.2.1.2> to provide effective interaction, our GUI design should use conventions, which are capable to increase a certain level of familiarity for users (Bickford, 1997). For example, the visual modelling environment should allow users to interact with the system by using see-and-point. In other words, the modelling environment should avoid manual data entry, use automatic data entry when it is applicable (Bickford, 1997). The see-and-point approach also can reduce the human memory related mistakes.
- <3.4.2.1.3> the visual modelling environment should allow users to be in charge (Bickford, 1997). The users should be able to manipulate their work whenever and whatever they want. In other words, what they see is what they get.

3.4.2.2. Usability Requirements

According to ISO 9241-11 (International Organization for Standardization, n.d.), it describes usability as: "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use". Nielsen (1993) outlines five aspects of usability, which includes learnablity, Efficiency, Memorability, Few Errors and Users 'satisfaction. We used Nielsen's usability principles as our conceptual environment to identify our usability requirements. Based on our interviews and discussion with the stakeholders, below are some usability requirements that our tool should have:

- Learnability: the ease of learning the functionality and behaviour of our tool. <3.4.2.2.1> instead of relying on users' memory, the visual modelling environment should allow the users learn the behaviours of a few simple interface functions, and then apply that knowledge in different situations (Nielsen, 1993).
- Efficiency: the level of attainable productivity, once the user has learned the system (Nielsen, 1993). <3.4.2.2.2> the visual modelling environment should achieve this by allowing users to model multiple projects simultaneously and switch to any project at any time. Normally, supply chain network modelling is very complex and a supply chain network model could have many different alternatives. It is not practical to put all alternatives into one diagram. Also, a complete supply chain network model could

contain different subsystems and parts from different diagrams or from different projects. These alternatives or subsystems or sub parts could form different diagrams or different projects. Hence, during supply chain network modelling process, users should be able switch between different diagrams and different projects easily.

- Memorability: the ease of remembering the functionality, so that the casual user can return to the tool after a period of non-use, without needing to learn again how to use it (Nielsen, 1993). <3.4.2.2.3> the visual modelling environment should achieve it by using software conventions and keep the GUI consistent.
- Few errors: users could make mistakes or errors, the modelling environment should help users recover from their mistakes easily (Nielsen, 1993). <3.4.2.2.4> the visual modelling environment should achieve it by using proper feedback and correction system to tell users what errors and mistakes are.
- Users 'satisfaction: <3.4.2.2.5> users should find our tool pleasant to use (Nielsen, 1993). The bottom line of our visual languages and modelling environment should not make our users.

3.5. Summary

In this chapter, we provided a discussion for the requirements and specifications for our VL and its tool design. Supply network modelling is a very complex and complicated work. We could not list all detailed requirements here in a limited space. We have focussed on a number of aspects related to our VL and tool development directly: the visual language requirements, and the visual environment modelling requirements. These identified requirements are critical for our next stage of development. In the next chapter, we will present our development process.

Chapter 4. Software Engineering 2.0

4.1. Introduction

Globally Distributed Software Development (Hossain, Babar, & Hye-young, 2009) is becoming very popular in both commercial and academic areas (Paasivaara, et al., 2009). For companies, there are powerful motivations (e.g. productivity) for outsourcing or offshore contract at lower cost (Vax & Michaud, 2008). For academics, it is also common to cooperate with other academics globally.

To boost SME users to participant and collaborate in supply chain modelling, the EC-funded SUDDEN project proposed a novel concept model (Mehandjiev, et al., 2009). MaramaSUDDEN is a suite of visual language and support tool which facilitate these supply chain practitioners to use the domain model in supply chain networks planning, design and collaboration (John Hosking, et al., 2008). We formed a collaborative research team comprising UK-based SUDDEN model and repository developers and New Zealand-based DSVL and modelling tool developers to tackle this problem domain. The research project team comprised the author, two researchers and a research associate in New Zealand, a researcher in the UK and two research associates in the UK. The author and UK research associates needed to work closely and collaboratively together for much of this project. The UK research team is a part of the SUDDEN project team. They are also direct users of MaramaSUDDEN. The UK research is also the project owner, who has a cross-functional role as a SCRUM Master for the targeted industrial SMEs. And our SCRUM Master functions as a "liaison officer" between the users and the developers.

Since our research project and its associated tool development were globally distributed, we were facing many non-technological problems in conducting our research. For example, accuracy of requirements, communication, coordination relationships and culture differences of organizations. Many studies show these problems can prevent the distributed development from being successful (B. Cohen & Thias, 2009) (Kontio, et al., 2004; Therrien, 2008) (Hole & Moe, 2008). In a similar way, we faced same or similar problems for our distributed research project. To ensure the product quality and productivity, we felt we needed to use more efficient and effective approach to overcome these problems.

Distributed Agile (DA) and Web 2.0 applications are becoming popular for globally distributed software development (Dillon, Chang, & Wongthongtham, 2008; Richards, 2009; Sureshchandra & Shrinivasavadhani, 2008). Using them proved very intuitive. We found they were quite appropriate and useful to solve the problems we faced through Global Software Development (GSD). Therefore, we adopted and tailored them in an informal way (incorporating Scrum, XP and Web2.0) to meet our development requirements.

In this chapter, we report the lessons learnt from our project on global distributed tool development by applying both distributed agile and Web2.0 technologies through the whole development process. For using Distributed agile (DA) and Web 2.0 applications together in our process, we called it Software Engineering2.0 (SE2.0). Our primary purpose for using agile methods was to solve development problems and improve productivity. The purpose for using Web2.0 technologies was to enhance knowledge sharing and the effectiveness and efficiency of communication. We found by using both distributed agile and Web 2.0 applications (as collaboration platforms/channels), some major difficulties were eased. Additionally both knowledge interoperability and productivity for global software development has been improved.

4.2. Web2.0 Adds New Value

Our development team was located in two locations and two time zones. One is in Auckland, New Zealand; another is in Manchester, United Kingdom with an 11-12 hours time difference between the time zones. When one team is working, the other is normally sleeping. The major non-technical challenges are effectiveness of communication and project management.

Over the past few years, many studies show Web2.0 services, especially social networking applications, e.g. Facebook, have become the most popular ways for people to communicate and stay in touch (DiMicco, et al., 2008; Jian, Shijin, & Liyi, 2008; J. Li, Psarrou, Sanxing, & Yajing, 2009; Schneider, Feldmann, Krishnamurthy, & Willinger, 2009). Web2.0 applications provide people capacities to share and interact to each other in personal life and business environment (DiMicco, et al., 2008; Jian, et al., 2008), e.g. Wikis, blogs, social networks. Traditionally, machines are the centres of networking. In contrast, the concept of Web2.0 is allow any people to be the centre of their own networks (Pastore, 2008). Our

approach using Web2.0 based GSD aims to ensure every member of our development team can be a centre of our development process.

In our GSD environment, we created a professional social network. There are a range of social network applications which offer very similar features. We used Google groups because it is easy to use, and invited all team members/stakeholders to join this private social network. We used this online based Web2.0 social network tool as one of our primary communication and collaboration platforms to collaborate and manage our research work and associated prototype tool development.

The social network normally has an owner and a manager and the rest are members. Each of the members has an individual profile. If the software development team members are first time collaborators, they can start to know each by reading each other's profiles without physical meetings. For our project, we got to know some developers via reading their experience and background in their e-profiles, and never physically meet. All development issues are organized in a Web based wiki style knowledge management repository for easy search and access. The Web2.0 Wikis were also used to hold and manage developer notes and user manuals. Figure 15 shows some examples of Web2.0 applications used in our project. The top left of Figure 15 shows a centralized repository for all discussion notes; and the top right of Figure 15 shows a list of files which were held in the centralized Web2.0 platform; and the bottom of Figure 15 shows a wiki page used to hold our project information. Web2.0 technology-based, generated flash movies also helped the team members and users to learn quickly, which reduced the long learning curve. For example, we used a demo movie which was published online to show users how to install and configure our modelling tool. Traditionally, it takes users' longer time to read user manual. The constructive idea of using Web2.0 technologies for knowledge interoperation was definitely a highlight in our development process.

Figure 15. Examples of Web2.0 applications used in our project.

We feel the most important benefit of using the Web 2.0 based social network, however, was sharing our ideas and opinions quickly and efficiently across all team members. Traditionally, in a GSD context, developers need to send many development notes and files to each other which are easy to lose control with distributed teams are becoming confused. The group also allowed us to collect these conversations, files and comments in one place that could be reviewed throughout the life of the project, forming a shared knowledge repository.

By using social network tools, all important development files, e.g. design diagrams, were uploaded to the social network's web storage. They could be ordered either by users or by dates or topics. New ideas/issues were posted to a Web2.0 style blog area for sharing, opinions, feedbacks and discussions. New WebPages could be easily created for different issues. Most of our development collaboration were done through Web2.0 social network services, e.g. updates, trace issues. Different solutions were able to be quickly provided by different stakeholders. Also recommendations and feedbacks were rather efficiently provided and seen by all team members. All members of this social network (the stakeholders of the software development team) could access these Web2.0 based application and edit them easily. Traditionally, development teams need to arrange many meetings. By using Web2.0 based social network service and applications, we saved considerable time from unnecessary

meetings. We even saved time from just avoiding the need to plan physical meetings. Figure 16 indicates a various generic tools can be used in the GSD environment.

Figure 16. Various tools and their uses in our GSD context

We found some members of our research social network (the stakeholders) were more active in expressing their opinions and making contributions via the social network service than traditional ways. Some study shows online based social networks avoid some embarrassing social situations (McDonald, 2003). The social network approach is helpful to improve the trust among the distributed development teams, and reduce communication difficulties, e.g. culture barriers (Julsrud & Schiefloe, 2007) . Consequently, compared with the traditional ways, by using Web2.0 applications and technologies, we improved the effectiveness and efficiency of communications directly and improved productivity indirectly. For instance, documentation for the tool, they were easy to access, categorize and maintain by using Web2.0 applications in our GSD environment. The Web2.0 platform also improved transparency for the project development and ensured team members were actively involved.

In summary, Web2.0 applications and tools leverage the new Web based technologies to help software engineers to share knowledge and improve development practices. To a certain degree, they overcome the weakness of GSD and we believe Web2.0 will change future approaches to GSD. Web2.0 technologies adds new value to software engineering, by helping to improve the effective communication and project management, which both are the key elements of successful distributed agile methodologies.

4.3. Distributed Agile Methodology

The agile approach is one of most popular software development methods to improve productivity (Reifer, 2002) and it is suitable for small team on budget and tight deadlines (D. Cohen, et al., 2004). For these reasons, we have adopted this approach.

4.4. Challenges for classic agile practices in a new environment

We understood that close collaboration among team members, and between developers and customers is vital to the success of the software development (D. Cohen, et al., 2004; Korkala & Abrahamsson, 2007). When the development team is distributed, it increases project rick even failure due to a range of environmental factors and challenges e.g. communication, (B. Cohen & Thias, 2009) and trust (Therrien, 2008). B. Cohen & Thias, 2009). B. Cohen & Thias (2009) expose the weakness and failure of distributed development by a number of case studies. B. Cohen & Thias (2009) also address "the effect of regular personal communications" is missing in distributed development due to the geographical distance.

Another common problem for software development is that developers don't understand the problem domain, making it difficult to communicate with other stakeholders. Stakeholders from different technical backgrounds use different jargons. The developers need to spend the time to learn and understand the problem domain. B. Cohen & Thias (2009) state that "it is much easier to consult with a co-worker on a question", when your co-work is physically around. The most productive way is to include stakeholders fully in the development process. The developers can learn the problems and requirements in person from the stakeholders. The GSD context, however, makes this approach very difficult.

For classic agile methodologies, face-to face communications and interpersonal relationships are key practices to ensure success of the project (Korkala & Abrahamsson, 2007) (D. Cohen, et al., 2004). For distributed software development, many classic agile practices become impossible, due to the difference of locations and the difference of time zones**,** e.g. pair programming. Thus, in the context of global development environment, some classic agile practices do not work.

4.5. Process and Practices

To cope with the challenges of our project and keep the project on time and on budget, we adapted and tailored a distributed SCRUM approach (Hossain, Babar, & Hye-young, 2009) to manage our development process with some adapted XP practices.

The reason we chose distributed SCRUM is that it suits our requirements for incremental system evolution. As we mentioned earlier, we identified major requirements and overviewed the outline of the tool at the early stage. After that we learnt about the detailed requirements at each iteration through the whole development lifecycle. Our development process iteratively evolves both requirements and the features under development by planning the development in short cycles (e.g. weekly or fortnightly). Since the process is incremental, it allowed our users to test our tool throughout the project development rather than waiting until the end of project. Their feedback was used to refine the features of our tool. We adopted UP (Arlow & Neustadt, 2005) as a primary conceptual framework to aid us for analysis, design and evaluation. We used distributed SCRUM to manage and organize our project. And user stories were discussed during the SCRUM meetings to summaries user requirements and Priority from the users' point of views. We used some XP practices (e.g. short release and customer review) to improve our productivity and quality of resulting modelling languages and tools. Figure 17 shows our development process. From SUDDEN supply chain models (1) we developed a set of candidate visual languages using a set of principled approaches (2).

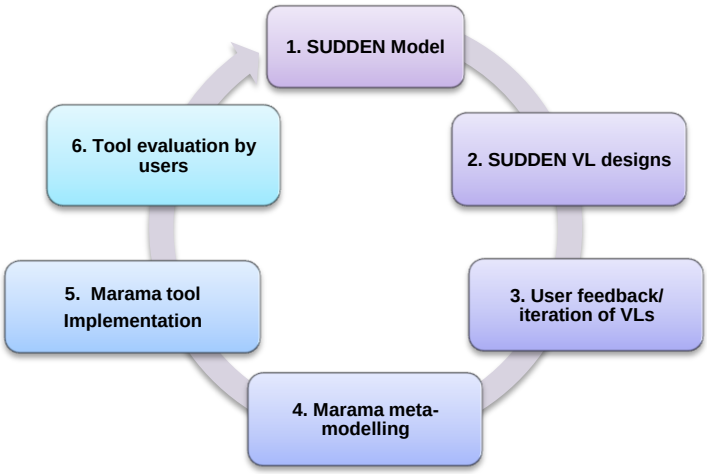

Figure 17. Our development process.

We gained user feedback on candidate VLs (3) and then developed a set of meta-models in our Marama visual modelling meta-tools to describe the VL elements and underlying SUDDEN constructs (4). We then developed a visual modelling tool using the Marama meta-tools, MaramaSUDDEN, including support for import and export of SUDDEN models to a shared semantic web based data repository (5). We evaluated the prototype tool with our target SME end users (6) and fed back results into both our visual language designs, design approach and underlying SUDDEN modelling concepts.

Compared with typical development GSD projects using SCRUM, we were facing the same range of risks at the beginning of our project e.g. the significant time zone difference, communication, "lack of tool support" and "lack of collaborative office environment" (Hossain, Babar, Hye-young, & Verner, 2009). According Hossain, Babar, Hye-young, et al.(2009), if a GSD project using SCRUM fails, the cause of the failure will fall into at least one of seven risk categories, e.g. lack of tool support. However, it is still a research issue if "SCRUM can be successfully used" in the GSD context (Hossain, Babar, & Hye-young, 2009). And it still requires empirical knowledge proof (Hossain, Babar, & Hye-young, 2009).

For our project, we did not adopt some recommended SCRUM practices. For example, we did not use "daily SCRUM meetings", instead weekly meeting to reduce the overhead of communication e.g. frequency and length (Hossain, Babar, Hye-young, et al., 2009). Due to 12 hours time zone difference at the course of the project, the pain of late night and early morning meetings was mitigated by eliminating daily SCRUM meetings and reducing the number of meetings. By highlighting using natural Internet resources (Web2.0 applications) in our GSD development, the most of risk factors were reduced, e.g. tool support,

communication. Our research exploration exercise used the iterative and incremental development framework (Hossain, Babar, Hye-young, et al., 2009), which is no difference from any other projects using SCRUM. But we used a number of online based Web2.0 applications to mitigate the pain from the significant time zone differences and geographical distance.

Below are some experiences from using Distributed Agile for our tool development in GSD environment.

1. **Multi-Channel Collaboration**

To improve the effectiveness of communications, we did not limit ourselves to the traditional channels. We believed that the best distributed agile practices will explore new technologies and approaches to keep team members connected.

We tried different methods to facilitate collaboration and communication, e.g. Instant Messaging, Emails, Internet Conference. Web2.0 is our primary channel to aid our collaboration. We used Web2.0 applications to manage the project and collaboration, which includes Web2.0 social networks and wikis.

As mentioned earlier, we set up a Web2.0 virtual collaboration working environment by using Google Groups as a knowledge sharing and project management collaboration environment.

By using this multi-channel collaboration environment, our team members can easily check and update project status, share knowledge and post and trace issues at anytime and anywhere. The source code of sub versions and daily releases with unified version numbers were kept persistently in separate online storage to ensure the development was consistent during the development period. Also, all developers used common code styles. All information (posts, backlogs and activities) of our social network and the project wikis were recorded persistently. Therefore, new comers could easily pickup and keep up with the project. By doing this, all tasks, activities and product issues could be easily managed, tracked and traced. Also we felt the multi-channel environment brings the remote users onsite. However, it is important to that all stakeholders agree on using the primary collaboration platform, a centralised knowledge repository and a backup solution.

2. Online based Scrum meetings

Throughout the four month development lifecycle, a typical Sprint was about one to two weeks duration depending on the work commitments and the development plan. Instead of having separate Sprint Planning, Review meetings, our Sprint meetings were multi-functional, which is more effective than the classic way when there is a significant time zone difference among the development teams. Weekly SCRUM meetings were about 30min-60mins using online conference applications, normally hosted by the Project Master. For each of the SCRUM meetings, we followed the same meeting structure. Firstly, we briefly updated the status and the progress of the development to team members (about 15mins). Secondly, stakeholders reviewed the previous work and provided feedback and opinion (about 15mins). Thirdly, at a product backlog session, team members identified the requirements and priorities and plan the tasks for the next Sprint (about 15mins). Table3 shows some user story examples of our backlog. They were identified during our SCRUM meetings. In Table 3, it contains four user stories for two different Sprints. Each of the user stories has some attributes including: a Task ID, a description, Priority ranking, assigned to and status.

Sprint I				
Task ID	Requirement Description	Priority Ranking	Assigned To	Status
1	As a target end user, I want to see the visual metaphors for all SUDDEN concepts so that I can use them to model supply chain networks directly; as a target end user.	must have	Rick and Cesar	Resolved
2	As a target end user, I want to have task-specific modelling interface so that I can use different sub-systems of the SUDDEN model respectively and	must have	Rick and Cesar	Resolved
Sprint II				
3	As a target end user, I want to have a database connection function so that I can choose any ontology database to work with. Priority ranking.	must have	Rick and Cesar	Active

4	As a target end user, I want to have a basic database explorer function so that I can browse the existing classes and instances in the database.	Could have	Rick and Chuy	Pending

Table 3. User story examples from our backlog.

Finally the last session was free discussion. By using this time, team members could get to know each other and provide informal opinions.

3. "Small and Simple" is important in GSD

We believe the easier, the better. We tried to avoid complicated tools. We used simple tools to make the development easier for team members. We prioritized the requirements and tackled them one by one. We also made sure we did not over commit ourselves for each of Sprints, ensuring each task was small and achievable. The product features and functions were developed one by one based on the SCRUM priorities. We used short time releases over mountains of documentation, and we kept simple and effective release notes only.

4.6. Discussion

Compare our research project with other projects (e.g. a development project), our project focused on the methodology study (e.g. user feedback or empirical evidence collection) to provide visualisation support for supply chain modelling, instead of just developing a software tool. Moreover, our direct users were cross functional (they had roles in both the SUDDEN project and the MaramaSUDDEN project). Since the SUDDEN project was developing in the course of our MaramaSUDDEN development, the requirements and priorities for our DSVLs and their support tool also evolved accordingly.

Apart from our main research on support for supply chain modelling, the difficulties of our GSD project lead us to the preliminary SE2.0 study by using our observations and interviews during the development process, which is more like a "bonus" for us. For the concept of SE2.0, unlike most of traditional research projects which make a hypothesis first before conducting any experiments or surveys, we used a bottom-up method to form it. Based on our literature review, we noticed there is a research gap for applying both Web2.0 and Distributed

Agile for GSD projects. We organized our observations first after our GSD project is complete, then we formed the SE 2.0 framework. Our method to form SE2.0 is very similar to the grounded theory (Sarker, Lau, & Sahay, 2000) in terms of the research approach. The purpose of using SE2.0 is to mitigate the challenges and risks of our GSD. Our research in this field is only a basic work. We believe our findings can be useful for further research.

There were a few things worked well by using this approach. Firstly, every stakeholders of our GSD project are familiar the Web2.0 applications used in our project. In fact, they have been using most of these Web2.0 applications in daily basis e.g. social networks. And these Web2.0 applications are open, free, easy to access and easy to use. There is no need for any training and learning to use these applications. Secondly, the SE2.0 framework is a "all-purpose" solution, which facilitated us in a number of areas of our development project e.g. discussions, knowledge sharing. And some major risks of GSD projects were mitigated more or less, which include "Asynchronous", "Lack of Group Awareness", "Poor Communication Bandwidth", Lack of Tool Support", "Lack of Collaborative Office Environment, Increased Number of Sites" (Hossain, Babar, Hye-young, et al., 2009). Finally but not least, apart from improving the effectiveness of team communication, it also improved the project management. For example, every stakeholder knew the updated status of our project by receiving messages or accessing the centralised project repositories.

However, there were a couple things our approach did not handle very well. Firstly, the SE2.0 framework heavily relies on the availability of the Internet, e.g. stakeholders must have steady Internet accesses to stay connected. We have experienced a few online meetings with poor output due to the unstable Internet access. Since some personals with intermitted Internet access, they hardly heard anything or only could catch a few words during these meetings. Therefore, the meetings need to be cancelled or re-scheduled. Secondly, it heavily relies on the Web2.0 application providers. During the life of our project, we also experienced the down time from our centralised online repositories. We could not get the latest updates from the online repository or the developer who was offline and sleeping. Afterwards, we also employed a backup method. If the development project is mission critical, using multiple repositories and backups is a must practice for SE2.0. Thirdly, big difference of time zones is still an issue for in-time user feedback. For our project, it is impossible to obtain in-time feedback from our users since they were normally sleeping in a different time zone. We needed to wait another 12-24 hours to get their feedback. Before that, it could be totally wrong decision we made during this 12 hours time. Finally but not

least, information security is also a concern. We did not experience any security problem during the life of our project, but it is a common concern for any Internet related projects.

In short SE2.0 would not cope with all challenges of GSD. But the SE2.0 framework can ease some difficulties and risks for some of GSD. As the environmental factors change, SE2.0 cannot guaranty 100% success for any GSD projects. It still requires lots of empirical study to proof if SE2.0 can successfully be used for most of GSD projects.

4.7. Conclusion and Future Work

In this chapter, we discussed the distributed agile and Web 2.0 we adopted and adapted in our tool development project. Through the whole development lifecycle, the distributed development team did not have any physical team gathering. Except from a preliminary meeting at the beginning of the project, most of team members did not meet face-to-face during the life of the project. If the project owner could meet the whole team a few times or we could have some group gathering activities, which would be much helpful for the group awareness and trust (Hossain, Babar, Hye-young, et al., 2009). And most of our discussion and feedback were asynchronous (Hossain, Babar, Hye-young, et al., 2009) via text documents and diagrams during the development process. It took some time to read and digest, which was time consuming. If most of user feedback could be recorded by video files and published on our private social network, it could save lots of time from doing lots of readings or misunderstanding. Although if agile should be used for GSD projects is still uncertain (Hossain, Babar, & Hye-young, 2009) , our SE2.0 approach facilitated us to overcome most of difficulties caused by the nature of our GSD project, which is not by chance.

For the future work, we need to collect more empirical evidence to find out what kind of GSD projects are suitable for using SE2.0. One single case study cannot deliver a empirically proofed result. Also we also need to identity all uncertain factors which are dependent on chance for using SE2.0. Moreover, both qualitative and quantitative researches are required to study the impact of the Web2.0 resources for different types of projects using distributed agile (e.g. a partnership project, an outsourcing project).

In summary, as the global market is becoming more internationalised, we feel GSD will become more popular. The Web2.0 technologies can help these projects to overcome many

GSD challenges as they assisted our development. For our project development, the Web2.0 applications indeed aided to relieve the pain of the distributed development. The SE 2.0 methodologies brought us a positive result for our GSD, in terms of project management, communication, productivity, interoperability of knowledge.

Chapter 5. Visual Language Design

5.1. Introduction

In chapter 3, we elaborated the requirements and specifications for MaramaSUDDEN. In this chapter, we present the visual notation design for the MaramaSUDDEN visual languages.

Moody points out the good visual language (VL) design should meet both users' perceptual and cognitive satisfaction (Moody, 2009a). One basic goal for all VL design is to enhance human-computer and human-human communication (Johannsen, 2009). Traditionally VL design involves lots of ad-hoc or arbitrary, e.g. colour selection (Bailey, Manktelow, & Olomolaiye, 2003) (Moretti & Lyons, 2005). And Moody states the visual notation design has been neglected by the software engineering community for many years (Moody, 2009b) .

To design a VL for the SUDDEN model, we took a scientific approach which includes carefully chosen metaphors and paradigms (Moody, 2009a). Our visual notation design was informed and guided by both the CDs (A. Blackwell, 2008) and Physics of Notations frameworks (Moody, 2009a). To enhance the usability of our DSVLs, we put more emphasis on VL prototyping methodologies for rapid visual notation design (e.g. visual elements and dimensions selection) and target end user feedback on using our prototype VL design on SUDDEN modelling problems. In this chapter, we discuss and address design issues for perceptual processing and cognitive processing (visual syntax and visual semantics) of visual notation design for complex abstract concept models. After that, we discuss the design of the meta-model of MaramaSUDDEN which supports our visual notations.

In this chapter, we separate our work into two sections. Firstly, we present a preliminary design with the visual notation design challenges. Secondly, we discuss our visual language design exercise for the SUDDEN concepts, which includes the visual icon design and the meta-model design.

5.2. Challenges and a rough Design

One of our primary goals for the design of MaramaSUDDEN is to facilitate users to perform domain specific supply chain modelling tasks visually and easily. For this goal, there are two major design tasks that need to be achieved. The first task is to design a set of visual metaphors and notations that allow users to visually model supply chains and which can express users' conceptual models into computer simulated models. The second task is to design a visual modelling environment to support these tasks and visual modelling languages, which will be discussed in Chapter 6.

In this section, we provide a brief discussion the challenges our visual notation design, a simplified version of our visual notation design. This part of work will be discussed in the next chapter.

5.2.1 Challenges

To design a suite of appropriate visual notations for the SUDDEN model, we faced a number challenges, including:

- **The SUDDEN model comprises a large set of domain-specific concepts** (John Hosking, et al., 2008) – each of these concepts needs to be visualized properly and rapidly. As we mentioned in the requirement analysis chapter, the visual symbols for the SUDDEN concepts not only need to be consistent, but also need to be distinguished from one another as well.
- **Abstraction** – most of these SUDDEN concepts are relatively abstract (John Hosking, et al., 2008). Abstract concepts do not have a specific form to present meanings (Myers & Baniassad, 2009). For example, the concept of "Process" of the SUDDEN model does not have a concrete representational form.
- **The SUDDEN model has many cross-cutting relationships** (John Hosking, et al., 2008), which are hard to express and understand.
- **The dynamics of supply chain network** requires that the visual metaphors of the MaramaSUDDEN model are maintainable, sustainable and evolvable and suited to the target end users modelling needs.

To overcome these challenges, we decided to create a set of scientific and formulised visual notation design paradigms with a range of basic design elements. These should allow us and other visual language developers to create the domain specific visual metaphors and notations rapidly and easily.

5.2.2 Design

To convert the SUDDEN concept model into practice, we believe there is no single powerful modelling notation to solve all these requirements and concerns; instead we adopted multiple visual notations.

Our visual notation paradigm design was based on both Physics of Notations and CDs frameworks. We set a few criteria for the design elements selection. Firstly, they must be familiar to everyone. Secondly, they are easy to be reproduced in different media. In addition, we mainly used the principles of Physics of notations to guide our design practices, and used the CDs framework to perform a continual evaluation for the notation design.

Our pre-defined visual notation paradigm includes a set of visual elements to express meanings, communicate between symbols and users, and interact among the visual symbols. The visual elements include a primary element shape, secondary notation elements (colours and textures), lines (have good characteristics to provide variations), and texts (provide direct meaning for visual symbols). All these design elements have a range of values to provide design capacities for our visual symbol. We assembled these design elements with distinguishable variations to express different meanings. If design requirements change, we can upgrade the existing notations, or create new ones easily by using the pre-defined paradigm.

5.3. Visual Language Design

5.3.1 Visual Metaphor Design Paradigm

The surface notation design for visual languages is a very challenging task that involves many concerns (Moody, 2009b). In most visual language design cases, the surface notations are used to convey concepts and messages in the interfaces (Moody, 2009a). Generally speaking, for the visual notation design, if there is no commonly recognized metaphor available for the domain specific concepts and objects, to design very sound and ideal surface notations can be very intricate and time consuming (Da-Qian & Kang, 1998).

As described earlier, the SUDDEN concept model has several dozen of concepts with a large number of relationship types between these concepts. These SUDDEN concepts and relationships are building blocks for the SUDDEN model. We surveyed existing research to see if there is any available notation in the supply chain domain can be used for the SUDDEN concepts. If so, they could be worthwhile for reference and could be used as a starting point for our design work. After the research, we found the existing icons cannot represent the SUDDEN concepts properly since the SUDDEN concepts are relatively abstract. Therefore, we defined a visual notation paradigm with a collection of selected design elements. We present a discussion for details of this paradigm below.

5.4. The elements of the visual notation paradigm

5.4.1 Generic Geometric Shapes

A shape is a certain area defined or enclosed by some closed boundaries. A shape has a variety of properties to express different meanings and communicate with users, and boundary is the most distinguish property (Myers & Baniassad, 2009). The types of shapes can be categorized in different ways, e.g. geometric shapes and natural shapes.

Why Shapes? – Moody highlights “shape plays a special role in discriminating between symbols as it represents the primary basis on which we identify objects in the real world”, and “shape should be used as the primary visual variable for distinguishing” (Moody, 2009a). Therefore, we decided to use generic geometric shapes as a primary design element, rather than any other shape types, as the basic design elements of our visual notation design templates since the SUDDEN concepts are rather abstract. Furthermore, geometric shapes are very maintainable and scalable. In contrast, other types of shapes, such as iconic forms, carry additional semantic meaning. For example, clubs, diamonds, hearts, and spades, are natural

shape types that could lead people to think about cards. A new moon shape could stand for the concept of Islam.

Capacities of shapes – boundaries of a shape is a good variation property, which provides opportunities for users to "divide the substance of our perception into distinct objects" (Myers & Baniassad, 2009). To make the appearance of shapes richer, we decided to use different line values (e.g. dotted line, solid line, colour of line) for the boundaries of shapes to enhance distinction. By using this technique, we created a range of basic symbolic templates. In addition, the scalability of shapes is another important property for our visual notation paradigm. The uniform size of shapes can improve the users' familiarity by increasing the consistency of visual icons. Gough, Green, & Billinghurs suggest (2006) "users become familiar with the interface, less cognitive effort is required to express their desires".

Usage in our VL design - the SUDDEN model has many concepts. For the UML 2.0 symbols are not sufficient and efficient for our domain specific visual language design. For instance, Moody points out using colour for visual notations in UML are forbidden. We decided to use a number of shapes to represent different SUDDEN classes with distinctive variations. We selected a range of generic shapes (e.g. rectangles,) to start with the visual notation design paradigm. Figure 18 shows a number of shapes selected for our design paradigm. These shapes include rectangle, round rectangle, oval, rhomb, triangles and so on. We believe these shapes are very easy to replicate and to remember either on computer screens or on other media like paper or whiteboards.

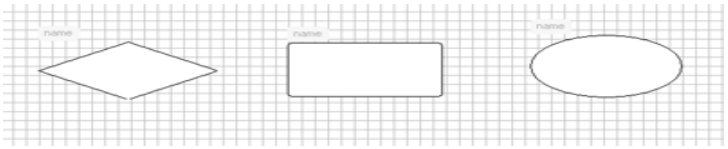

Figure 18. Generic shapes selected for our visual notation design paradigm

5.4.2 Colours

Colour is a reflection of light on the surface of objects. Colours can be also classified by different criteria, for example, warm colours and cool colours. For example, in common usage, red and yellow are warm colours; and white and blue are cool colours.

Why Colours? –Moody suggests colours have significant impact to our brains via our eyes and nerve systems (Moody, 2009a). In other words, colours can affect our nerve and

emotional systems dramatically. Traditionally, colours have been used as one of the primary elements for graphical design. Colours can help people to remember and can catch their attention (Moody, 2009a). A good example of using colours for notation design is the traffic lights and traffic signs systems. A red colour means "stop", no matter if it is a red traffic light or a red traffic sign.

Trade-offs - Moody's study shows the disadvantages of using colours for visual notation design (Moody, 2009a). For example, if the notations are printed out in grey scale, all non-white colours would automatically become grey. Furthermore, if someone is colour-blind, colours can be useless to convey messages. Also, for different cultures and communities, different colour systems have different meanings. Some selection of the colour themes could be confusing for different users since different people perceive colour differently. Traditionally, colour selection for visual notation is rather ad-hoc (Bailey, et al., 2003). Moody's research indicates colour is still one of best visual design elements to produce visual effect for visual language design. Therefore, we decided to use colours as secondary notations (Green & Petre, 1996) for our visual notation design.

Capacities - colour value is an important characteristics of colours (Moretti & Lyons, 2005). For our graphic notation design, we can change the value of colour to make the related concepts consistent, and capable to keep non-relative concepts distinguish to each other.

Usage in MaramaSUDDEN design - we did not want to choose colour for visual notations randomly. Our approach is to find the best matching colour for each of the core SUDDEN concepts first. For example, "Value Creation" is a core domain concept which has a number of child concepts and associated concepts. For this case, we found "blue" is the best matching colour for "Value Creation". We chose blue as its secondary notation colour because blue generally stands for "new idea" and "new adventure" or "high-tech" in common usage, e.g. blue ocean stands for innovation (Burke, Stel, Thurik, & Revision, 2009). The evaluation for selected colour themes is performed by our direct users.

A colour theme involves using a combination of colours for backgrounds, lines and shapes. Our colour selection of the template design focused on the balance of the diversity and integrity to the visual notations as a whole. After that, this selected colour becomes the colour theme for the family members of "Value Creation". We used clear variations of this colour to match the child classes and associated classes of "Value Creation" i.e. these children and associates are coloured a variation of blued used to colour Value Creation classes.

We put emphasis on the user experience of using different colour themes in visual notation design, especially the readability and the memorizability. How best is it to allow users to easily understand the notations and remember them? We found simplification to be the best way to achieve this. However, when the number of classes increased, the harder the simplification task became. In our case, we had to restrict ourselves to a limited number of colours, in other words, reducing the number and frequency of colours used for each of the visual notation templates. Figure 19 shows a number of colour themes selected.

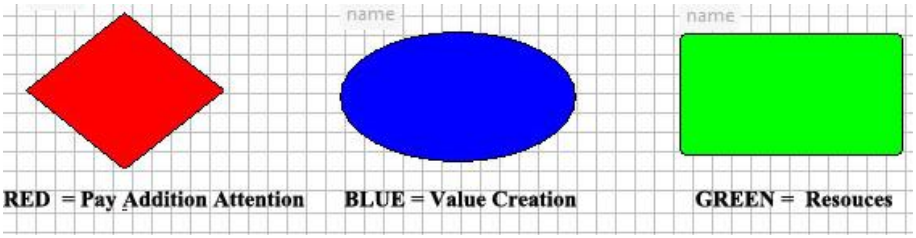

Figure 19. Theme colours selected for our design paradigm

5.4.3 Symbolic textures

Texture means the surface of objects. In a physical term, it is the nature of materials, e.g. paper or cloth. If it is a visual texture, people only can see if, but cannot touch or feel it. There are different styles of textures and different ways to use textures in graphic design, e.g. warm versus cool.

Why Texture? - Symbolic textures can enrich the capacities of the visual notation design templates. The basic function of using these textures in the visual notations is to allow users distinguish them in alternative ways when colours do not function in some scenarios. We believe they can improve the overall user experience and enhance the appearance of the visual notations.

Capacities – a symbolic texture is a combination of colours, shapes, space, text and lines. All these components are good features to express meaningful concepts.

Usage in MaramaSUDDEN design – Our concept of using textures in our DSVLs is to use texture as secondary notations. Our philosophy is to use unique visual textures for each of visual notations, but keep it as simple as possible. However, it is very hard to interpret each of the SUDDEN concepts into a very sound texture in the given design time-frames.

Therefore, the textures in MaramaSUDDEN are only used to provide clear distinctions and variations.

To create textures for the visual notations, we used a mixture of available visual components, e.g. patterns, space, layers, colours, and lines. All these components assemble a symbolic texture, which delivers "styles" for our visual notations. In addition, we used strong colour contrast technique to stimuli users perceptual processing (Moody, 2009a). For example, we used both soft colour (grey silver) and bright colour (dark blue) within the same texture pattern.

To design high quality of textures, we tried to make related concepts consistent in the diagrams. Figure 20 shows some textures used in our visual notation design.

Figure 20. Texture patterns used for "LogisticService", "Node" and "Machine" to make these symbols more distinguishable.

Due to our time constraints, some textures used in our design are still very basic, and lack any distinct meanings. However, we believe the usage of the texture patterns can express different meanings very well. For us, using the texture for the notation design is still at an experimental stage, we still need to explore how to maximize the usage of texture while integrating with other design elements in visual notation design.

5.4.4 Line

A line is formed by numbers of dots. Lines are another essential element for our visual notation design. There are different ways to classify line types.

Why Lines? -we used lines as a fundamental design component for the textures, the shapes of the visual notations, and relationships. Lines combined with use of different head shapes (e.g. arrow or triangle) can guide users' eyes very well. Arrowed typed lines can thus have a dominant influence on users' perception. Also, combined with different shapes such as triangles or diamonds, Figure 21 shows the different combinations of lines and shapes can

represent different meanings. In addition, lines also can help to arrange computer screens and shapes.

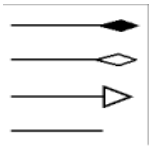

Figure 21. Visual representations for relationships

Capacities - as a design element, lines have some very good values, e.g. colour, stroke. Different styles of lines can represent different meanings. The direction of lines is a useful property too. A single straight line has two directions no matter if it is horizontal or vertical.

Usage in MaramaSUDDEN design - For our visual notation design, we used lines for both the class-type and relationship-type notations. For the classes, we used a range of line types e.g. dotted line, to enhance the appearance of the visual notations, which make the visual notations more eye-catching and clearer. We also used lines to texturize the surface notations and organize space of the diagrams. The line variations are used for our visual notations design are dotted, concrete, thick, and thin and colours, which make the classes and relationships are very distinguishing.

For the class type notations, we used lines to assist our texture design. Different line types could send users different message. For example, dotted lines around the shapes stand for a "possible/alternative" option. For Figure 22, the horizontal blue lines are very wide, and they are consistent and uniformed. These lines help us to create a peaceful texture, which is inspired by national flags, French flag.

Figure 22. Dotted lines used in our visual notation design.

For the SUDDEN relationships, our approach is to make sure the selected line types worked for our direct end users (the SUDDEN project team) who share the same domain and knowledge background. And it is important to make the visual metaphors readable and

identifiable. During relationship type symbol design, we found it is very useful to use the same line type pattern with strong variations, e.g. use some different colours or thickness.

In addition, we used a varied range of lines to define and describe the relationships combining with different shapes, e.g. in the form of an arrow. These specially designed relationships can influence the appearance of the whole network diagram. For the relationships, different relationship types need to be represented by different visual notations. We used different line directions to carry different semantic meanings.

For the SUDDEN model, many classes have N-nary relationships (John Hosking, et al., 2008). In another words, there can be more than one relationship existing between two classes. Using line direction can solve this issue. We used lines combining with shapes to convey different information. These combinations form different line patterns for the SUDDEN relationships. For example, Figure 23 shows a symbol for association types.

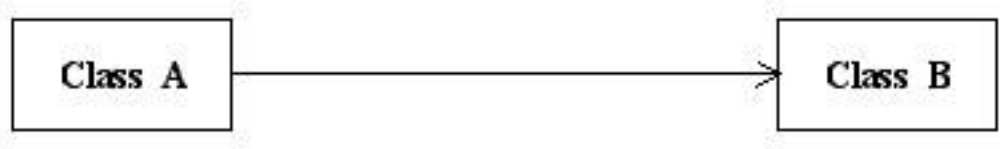

Figure 23. A symbol for association types used in our design.

However, as the number of relationships increased, this made notation design more difficult. For the entire diagram, we also need to consider integrity and unity of the whole interface. To avoid making the interface too busy, we only chose a few line templates to apply to the relationship type notation.

5.4.5 Text

Text is a primary visual dimension of the visual notations (N. Hari Narayanan & Roland, 1998).

Why Text? - Text makes the visual notations more readable and express clear meanings. We used text as the last design element for our visual notation design template.

Capacities - text has good visual properties, e.g. font style, and colour. It is natural for people to read meanings from words (Horn, 1999). Horn suggests it will not be adequate for icon design without using text. Horn states "only use text if the meaning of the icon is likely to be ambiguous" (Horn, 1999). Horn also points out that using text in icon design can prevent users from guessing (Horn, 1999).

Usage in our design - text is used in visual symbols to explain the name of the concepts. Some of the SUDDEN class names are very long. To make the visual notations well formed, we do not want the text of the visual notations to be too wordy. Therefore, we used some abbreviations for some words, for example, "rqmts" stands for "requirements". By using the abbreviations, we made the visual notations look better.

In addition, we used a strong contrast technique to make these text elements stand out. For example, if the background texture is dark blue for the visual notations, then we used silver grey for the text, which forms a strong contrast. But the whole visual notation still looks nice. We think it is a good practice that we always place the text on the top of the notations, so that users can see it immediately since human being's are trained to accept information from top to bottom (and left to right in western cultures).

5.5. MaramaSUDDEN Visual Notation Design

We needed to define a set of DSVLs for our MaramaSUDDEN tool. In our initial design, we took Representational Epistemology approach (Barone & Cheng, 2004) to provide a simple complex view (John Hosking, et al., 2008). However, as the SUDDEN modelling was still developing, there are more concepts being developed in the SUDDEN model. Consequently, all these concepts and relationships need to be visualised in MaramaSUDDEN. In the final version of the SUDDEN model, there are several dozens of concepts and relationships in the core domain model. We felt one complex view could not provide a proper modelling space for our users (John Hosking, et al., 2008). Therefore, we decided to split the domain into three different views. For the details how these visual symbols assemble a visual diagram to solve supply chain issues, please refer to chapter 6.

In previous section, we identified the visual elements of for our paradigm. In this section, we discuss the details of visual syntax and visual semantic design for MaramaSUDDEN. We put emphasis on both syntax and semantics of visual language design. Furthermore, both

consistency and variation have been addressed in our design. Each visual symbol of MaramaSUDDEN is assembled by various visual elements from our pre-defined paradigm, e.g. shape, colour. Next, we use ordered numbers to help our readers to find the corresponding classes and relationships in different figures.

5.5.1 Notation design for the Product DSVL

Figure 24 is a screen shot for the Product DSVL in use. The Product DSVL is to assist users to perform production decomposition modelling in supply chain networks. In the SUDDEN model, the "Product Decomposition", "Complex Part" and "Simple Part" are children of concept "Value Creation". In common usage, blue has been used to stand for "creative" or "new". Therefore, we used bluish colours as theme for this "Value Creation" family. For the MaramaSUDDEN classes, we mainly used rectangle and round rectangle with variations, e.g. thickness and different colours, which provides a feeling of familiarity to our users.

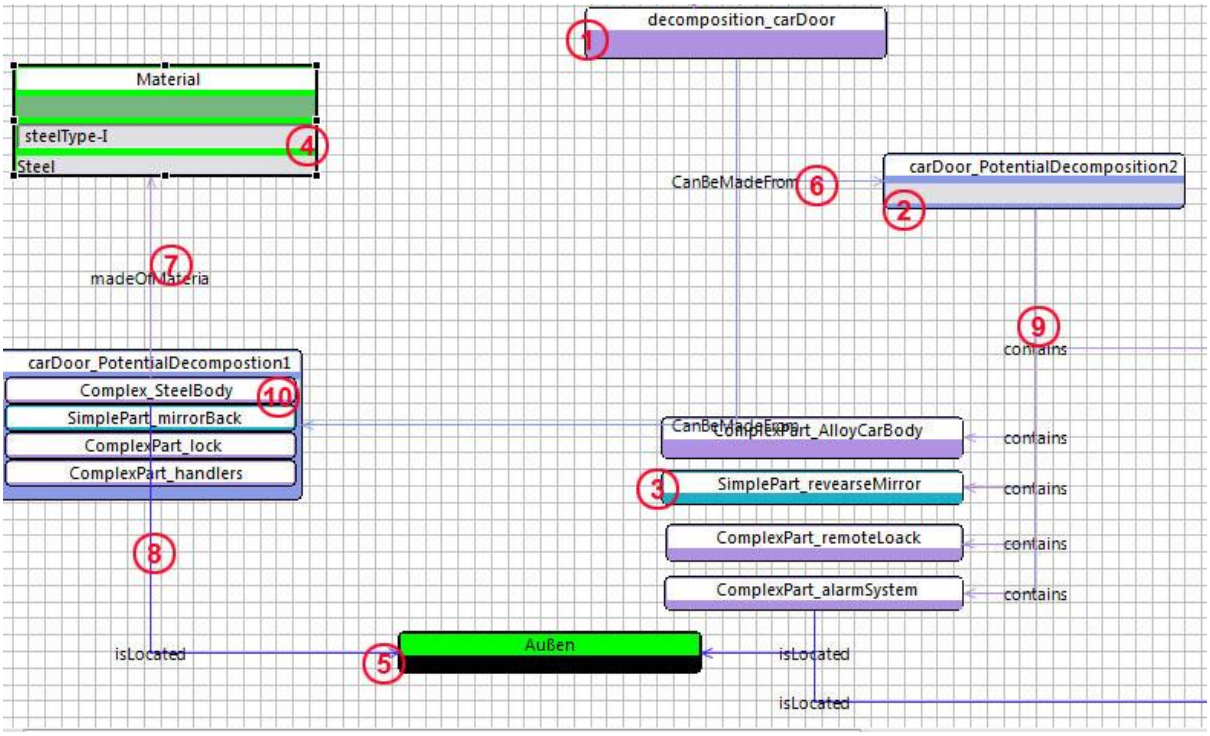

Figure 24. A screen shot for an example of the Product DSVL.

1. In Figure 24, No.1 shape is a “Complex Part” class. We used bluish purple and white to suit this symbol, which deliveries a message of “I am a child of Value Creation”. The strong contrast between bluish purple and white forms a symbolic texture that facilitates users to remember with some perceptual stimuli. The text phrases are located on top of the symbol which matches people’s reading conventions (from top-to-bottom in English culture).
2. No.2 shape is a “Potential Decompositions” class, and No.3 is a “Simple Part” class. They are also child classes of “Value Creation”. They share the same design concept with the “Complex Part” class (No.1 shape) with distinguishable colour variations. Users could tell they are part of “Value Creation” family via the colour theme used in the visual icons.
3. No.4 shape is a “Material” class. In the SUDDEN model, the “Material” is a child class of “Resources”. In common usage, green has been used to stand for “hope”, “future”, and “energy” and etc. Therefore, we selected green as a colour theme for the “Resources” family. And a symbolic texture pattern also used to distinguish it from No.5 shape, which is a “Location” class.
4. The No.5 is a “Location” class. We used green and black to form a strong contrast variation.
5. No. 6 is a “CanBeMadeFrom” relationship between a “Complex Part” (No. 1 shape) and a “Potential_Decomposition” (No. 2 shape). This visual symbol comprises a collection of elements which are a line, an arrow, colour and a functional textual phrase. The open arrow stands for the direction of the visual communication flow, and helps users to organise the diagram space. The text phrase is used to express the meanings of the relationships effectively. We used the bluish purple to indicate this relationship type connector is related to the “Complex Part” class. And it is also associated with the “Value Creation” family.
6. No 7, No 8, No 9 relationships share the same visual syntax of No. 6 connector. No 7 is a “made of” relationship between “Materials” and “Complex Parts”. No. 8 is a “located relationship” between “Complex Parts” and “Locations”. No. 9 is a “contains” relationship between a “Potential Decomposition” and “Complex Parts”.
7. No 10 shape is a modularized “Potential Decomposition” which contains a set of nested icons which are a number of “Complex Parts” and “Simple Parts”. We used vertical alignment layout conventions (John Hosking, et al., 2008) to encapsulate classes to save screen space. The nested shapes within this “Complex Part” module

can be easily identified by using colour contrast techniques. Also, "layout/placement constraints and conventions to provide distinctions" (John Hosking, et al., 2008).

5.5.2 Notation design for the Process DSVL

For notation design for the Process DSVL, we used three major colour themes with strong variations, which are green (for the "Resources" family), blue (for "Value Creation" family) and "Red" (for the classes require addition precautions). We also used a range of textures, shapes and lines that provides clear distinction for each of the sub-notations.

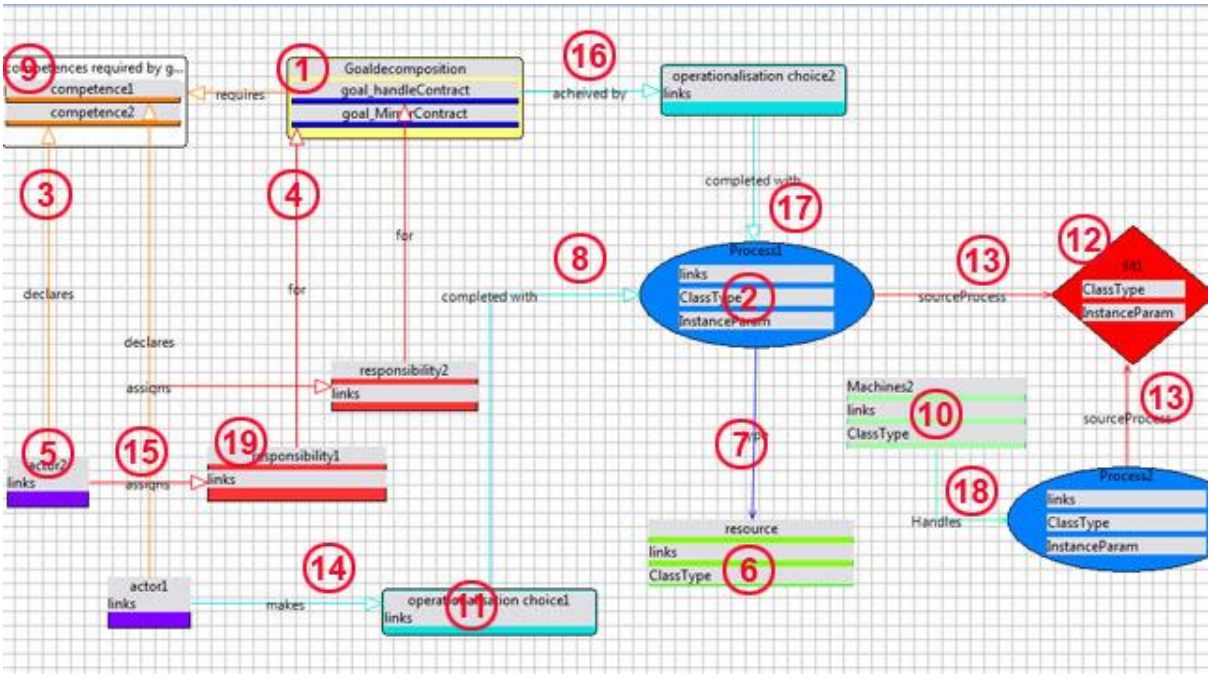

Figure 25. A screen shot for the Process DSVL.

1. In Figure 25, No.1 shape is a "Goal Decomposition" class and No.2 shape is a "Process" class. Both "Process" and "Goal" are child concepts of "Value Creation", which should follow the same blue colour theme. We used variations of shapes and colours to provide clear distinction in both visual syntax and visual semantics for child classes in the same family. For "Processes", we used an oval shape to represent the dynamics of processes. We used sky-blue for "Processes" and navy-blue for "Goals", and therefore they can be easily identified by users. The "Goal Decomposition: shape provides container for a decomposition of "Goals". As a shell

for decomposition of goals, the shape of "Goal Decomposition" only works in the background. We used yellow colour to create strong contrast to the navy-blue used for "Goals".

2. No.5 shape is an "Actor" class, which is relating to "Value Creation". Purple is a colour which closes to blue, hence we used purple to make "Actors" associate with the "Value Creation".
3. No. 6 shape is a "Resources" class. As we mentioned earlier, all "Resources" family used green and greenish colours. For example, No. 10 is a "Machine" class, which is a child class of "Resources". We also used the green theme with a strong variation. Using a colour theme in our visual syntax provides consistency for the visual symbols of the same family.
4. No. 9 shape is a collection of "Competencies". No.19 shape is a "Responsibility" class. An "Actor" needs to have certain "Competencies" to obtain "Responsibilities" to make decisions for a certain "Goal". No.12 shape is a "fit" relationship class, which is one of basic process dependency types. (Malone, et al., 2003). For these three classes, we need to alert users to pay more attentions since these classes are critical to the process modelling. The visual icons need warn users and draw extra attentions. We used orange and red with variations to make them distinguish from the rest of icons. Both red and orange have been used for a number of warning systems. For example, for US homeland security, orange is used to stand for "taking additional precautions", and red stands for "critical emergency" (US Homeland Security, 2008). For the "fit" concept, we also used a distinguishable Rhomb shape.
5. No. 11 shape is an "operationalisation choice" class, which is also relating to "Value Creation" family. We used cyan, which is between blue and green.
6. The relationships between different classes inherit the colour theme from the classes they are associated. For example, No.8 relationship uses cyan and an arrow to represent the meaning "completed-by", since an "operational choice" is "completed-by" a "Process". The arrow also used to guide users eyes to obtain the meaning of the visual sentence (Moody, 2009a). No.3, No.4, No.14, No.15, N0.16 and No.18 share the same design syntax of No. 8 connector.

5.5.3 Notation design for the ASN DSVL

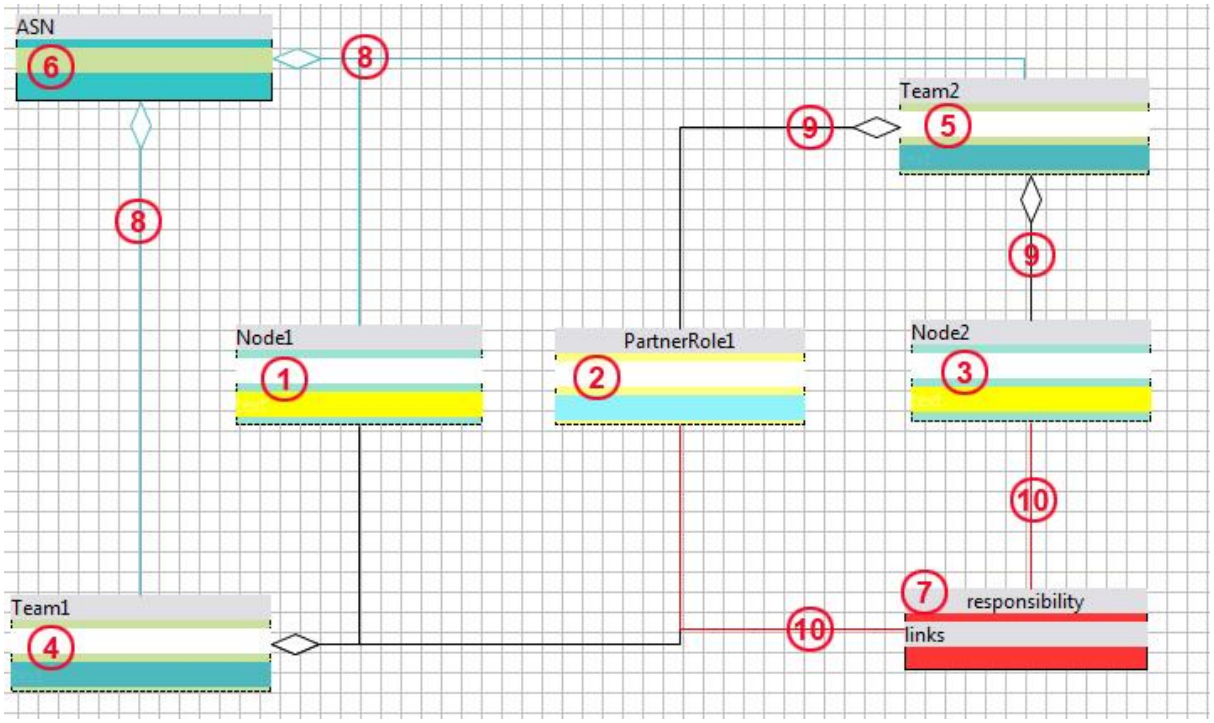

Figure 26. A screen shot for the ASN DSVL.

1. In Figure 26, No.1 and No.3 shapes are "Nodes" of virtual enterprises (Mehandjiev, et al., 2009). A "Node" could be a person, or a company. To form a supply chain team (No.4 and No.5 shapes), Nodes need to find other suppliers (Partner Role), which is No.2 shape. No.5 shape (team1) and No.4 shape (team2) forms an ASN (No.6 shape).
2. All "Nodes", "Partners", "Teams" are in the same family of ASN. We used iconic texture to keep these family members consistent. We also used dotted line for the boundaries of "Nodes", "Partners", and "Teams" icons to represent the dynamic changes of supply chain teams. No.7 shape is "Responsibility" class. Both "Nodes" and their "Partners" share "Responsibilities". This is same "Responsibilities" class that is the same class used in the Process view type.
3. The symbolic textures used in this sub-notation design are for experimenting with the impact of texture in visual notation design. Our symbolic texture design concepts are inspired by the national flags design, e.g. French National flag. We used a few basic colours and spaces to deliver "styles" of textures. At this stage, the texture design does not carry any significant semantic meanings.

5.6. Meta-Model Design

In this section, we present the design of the MaramaSUDDEN meta-models, which support the visual notations in terms of syntax and semantic behaviours. These syntax and rules of MaramaSUDDEN meta-models need to be consistent with the SUDDEN model. Since our implementation is based on using the Marama meta-tools, which support extended entity-relationship meta-models, our primary task is to convert the SUDDEN model from a concept model into extended entity-relationship meta-models.

First of all, we used numerous entity classes to represent the elements of the model underlying the visual notations. Because each of the visual elements needs a name as a unique identifier and a class type to describe where it comes from, we have created two standard properties for each of the entity classes. One is "name" as the key identifier, and another one is "Class Type". Some of the visual notations required extra properties. We designed these special properties case by case. These properties of the entity class describe the attributes of the visual notations. Figure 27 shows an entity class of the MaramaSUDDEN meta-models.

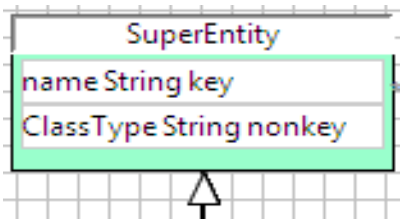

Figure 27. An entity class of the MaramaSUDDEN meta-model.

Our second task is to design the relationships among these entity classes, which would define and describe all relationships of the SUDDEN model, which are mapped to the visual notations as well. They define the abstract syntax of the relationships. Figure 28 indicates some examples of entity relationships in the MaramaSUDDEN meta-models. In Figure 28, Goal, Function and Part are the children of the super entity.

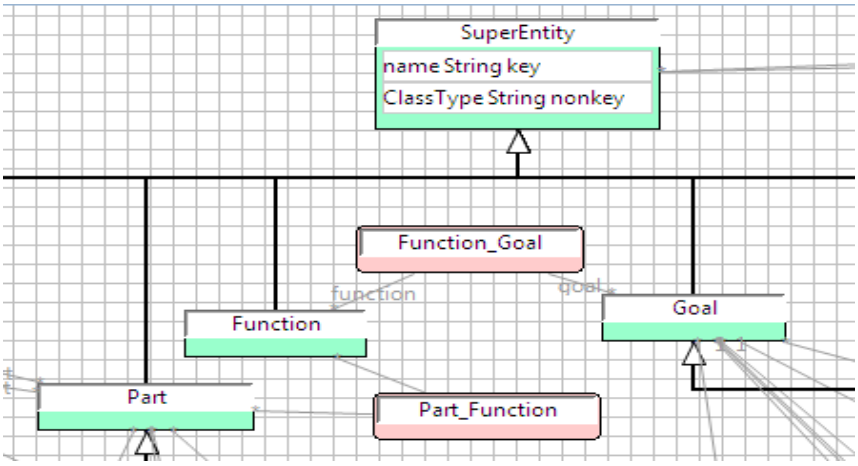

Figure 28. Examples of entity relationships in the MaramaSUDDEN meta-models.

In the SUDDEN model, there is often more than one relationship type between two concepts. To make sure the connectors appropriate carry semantic meanings, we used multiple relationships to describe and define these relationships in the meta-models of MaramaSUDDEN. Figure 29 indicates an example of multiple relationships in the meta-model. In Figure 29, between Part and ComplexPart, they are binary relationships.

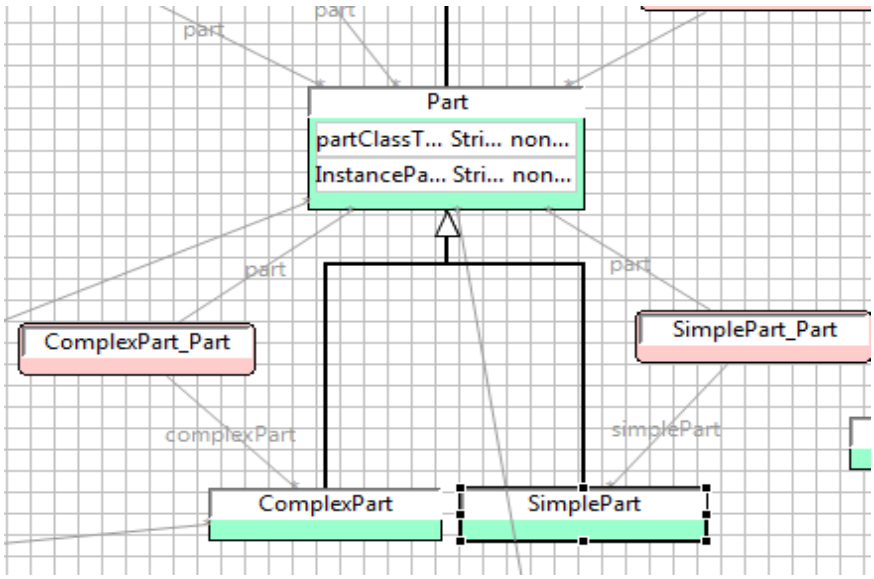

Figure 29. Multiple Entity Relationships: Between Part and Complex Part classes, there are two different relationships

In summary, our extended entity-relationship type meta-models include three sub-systems of the SUDDEN model, which are a process system, a product system and an abstract supply chain system. Figure 30 shows the extended entity-relationship (ER) based meta-models of MaramaSUDDEN.

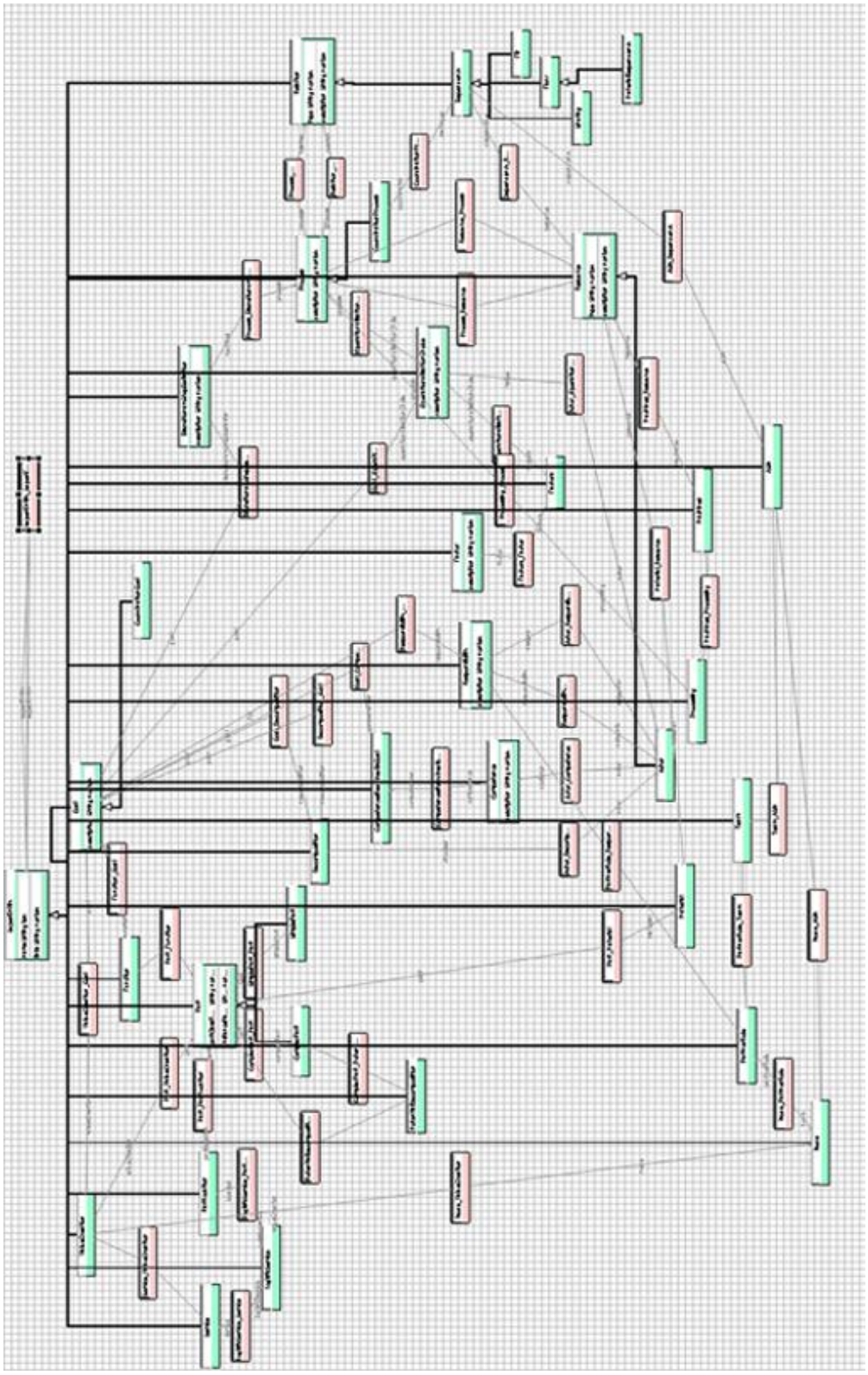

Figure 30. The Meta-models of MaramaSUDDEN

5.7. Summary

We felt a very good scientific visual notation design template should be a combination of science and art. However, Moretti & Lyons state “software developers rarely have expertise in graphic design or colour selection and will tend to use the default colours for widgets” (Moretti & Lyons, 2005).

Our surface notation design for the domain model was guided by both CDs and Physics of Notations frameworks; we selected a range of design elements to formalize a visual notation design paradigm for rapid notation creation. As a result of using the visual notation design paradigm, we were able to quickly and easily construct a large set of sophisticated domain-specific visual metaphors. We argue that these paradigm-driven visual notations could provide clarity in notations understand and save developers a lot of time in learning. With the support of these visual notations, the users of MaramaSUDDEN could easily understand the concepts of the SUDDEN model, and create and customize their own supply chain models efficiently.

Chapter 6. Modelling Environment Design

6.1. Introduction

In chapter 5, we presented the visual notation design for our supply chain modeling languages. In this chapter, we provide a detailed design for the MaramaSUDDEN modeling environment focusing on a model-driven architecture-based approach for supply chain modeling support. We also discuss the major design decisions we made, including the key mechanisms and techniques we selected for our tool.

We classify our modeling environment design work into six sections: 1) a discussion of the Marama tools, 2) an architectural design of MaramaSUDDEN, 3) an overview of different layers, 4) multi-view design, 5) interface design, and 6) mechanisms and functional design. Lastly, we summarize our work briefly. Due to resource constraints, we believe the MaramaSUDDEN system still has some drawbacks which will require additional work to rectify.

6.2. The Preliminary Design

For the architecture of our tool, we used a model-based multi-layer structure to separate the user interface, the data and the code modules into different layers. Grundy, Hosking, Nianping, & Na (2006) state a number of "graphical frameworks provide low level yet powerful sets of reusable facilities for building diagramming tools", e.g. MVC (Sanderson, 2008) and Eclipse GEF (*Eclipse development using the graphical editing framework and the eclipse modeling framework*, 2004). We assert that it is the best approach to keep and share knowledge, and ensure that data is portable for this domain. Based on our requirements analysis we developed a set of functions which form the basis of our tool. The key features of MaramaSUDDEN include sub-domain and multiple alternative views modeling support (John Hosking, et al., 2008), abstract hierarchy and modularization support (Moody, 2009a), domain-specific knowledge management support, consistency validation (syntax and semantics consistency) support, and other easy-to-use facilities.

MaramaSUDDEN is configurable and extendable and supports the addition of new functions and features. By using a view-based visual modeling approach, our users can design their supply chain model views, and save their work into a computer readable semantic web format easily. The user designed models also can be kept as reusable templates. This is specifically useful for visually editing multiple complex enterprise processes within customized domain specific supply chain models. We believe it is also easy to tailor MaramaSUDDEN to meet different requirements of supply chain modeling.

6.3. Modelling Environment Design

6.3.1 The Choice of Meta Tools

To rapidly develop our modelling environment, we explored different options, e.g. MetaEdit+ (Tolvanen & Kelly, 2009), Meta Builder (Ferguson, Hunter, & Hardy, 2000) , visual studio dsl tools (Cook, Jones, Kent, & Wills, 2007). After comparison, we decided to use the Marama framework for our support environment development. Marama is a set of Eclipse plug-ins that realizes domain-specific visual modeling tools specified using high-level DSVL tool specifications (Liu et al, 2007). Marama allows users to rapidly specify or modify a desired visual language tool using a high-quality Eclipse-based editing environment. Marama has many advantages for our tool development over others. We discuss the advantages of choosing the Marama meta-tools below.

First of all, Marama has been used for developing several visual languages locally at Auckland, e.g. MaramaEML - business process modelling language for modelling enterprise process (L. Li, Hosking, & Grundy, 2008). We could benefit from the experience from having the developers of Marama available and from their experience developing these visual languages.

Secondly, the Marama framework is totally open source and it is platform independent. This is very important for our tool development since users of our visual modelling tool could use different operating systems, e.g. Windows, Linux.

Finally, the Marama framework provides multi-view support for tool development. The SUDDEN concept has its own problem domains and sub domains. Each of these sub domains is responsible for specific tasks and responsibilities. These sub domains or sub systems of the

SUDDEN concept model needs to be defined and modelled in separate domain specific environments. Figure 31 shows the structure of the Marama meta-tools.

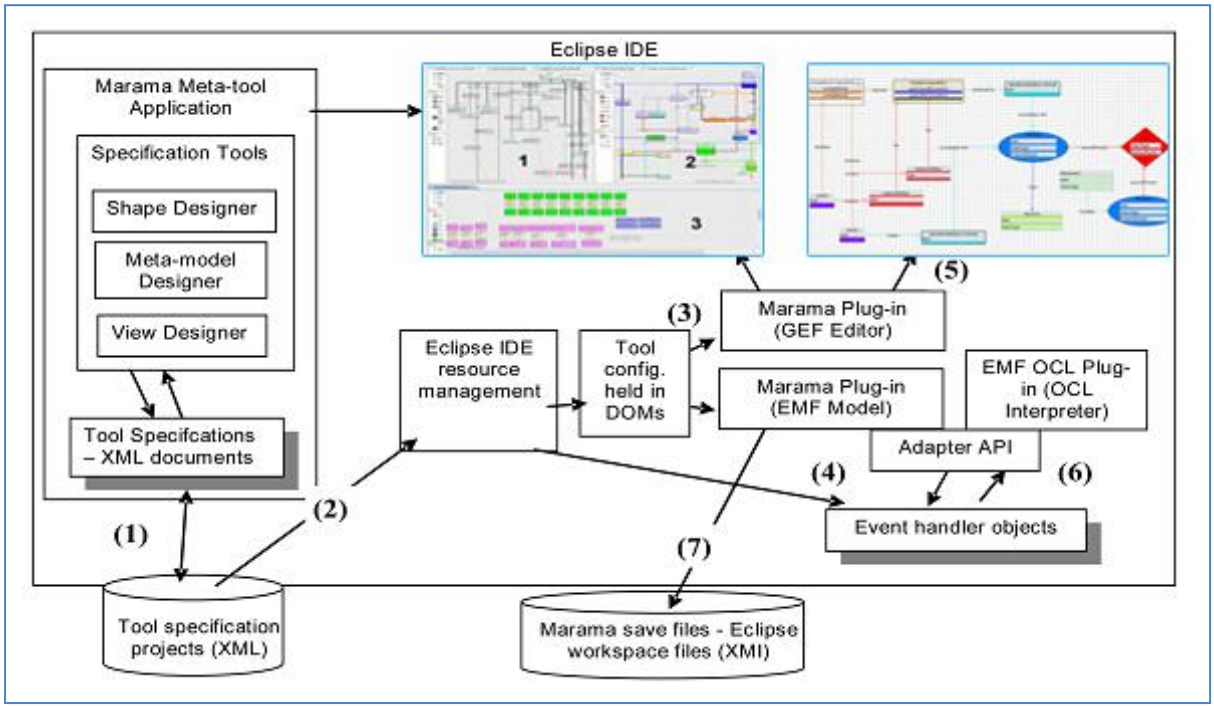

Figure 31. The structure of the Marama meta- tools.

Below are some major features of the Marama meta-tools.

- Provide meta-modelling support via a meta-model designer.
- Provide graphic notation support via a Shape designer.
- The meta-models and the surface notations can be mapped through a View Type designer.
- The system events can be handled and created at runtime.

6.3.2 Architecture design

To build up the visual modelling environment, we decided to use four-layer architecture. There are many benefits to using multi-layer design, e.g. security and reusability. The key reason to use a multi-layer design in our tool is to keep the data repositories logically independent and portable. And multiple applications can access and interact with the data repositories. Our four layer architecture includes a presentation layer, a business logic layer, a utilities layer and a data layer. Figure 32 indicates the four layers of our support tool.

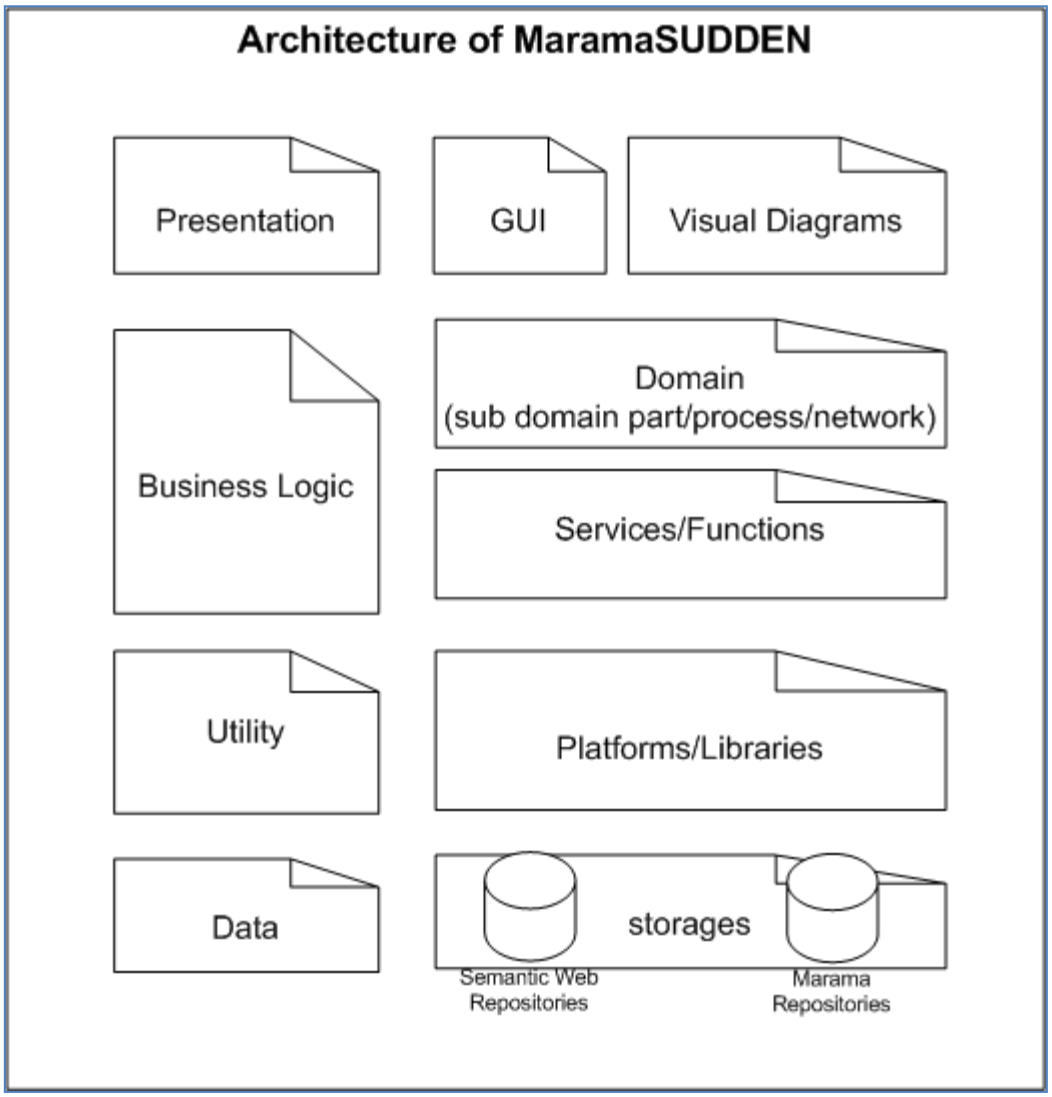

Figure 32. The four layers of the MaramaSUDDEN tool.

- **The presentation layer** is the visual interface of the modelling environment, which interacts with end users directly. This layer is responsible for obtaining users' input and decisions, and then sends the data to the next layer. It also shows information from the system to end users. The presentation layer comprises a view based visual modelling interface, GUI windows, messages, browsers and etc. The presentation layer is the most important layer for user experience and implementing the visual languages, since it works as a bridge between end users and the modelling environment.
- **The business layer** defines and describes the specific problem domain/sub-domains of the SUDDEN model. It also specifies the behaviours of the sub domains. For the SUDDEN model, there are three sub domains: Product, Process, and Abstract Supply

Network. The syntax and semantics of these three specific sub domains are decided and controlled by the business layer. To better serve the system requirements we decided to separate the business layer into two sub layers: a sub domain layer and a services layer. By doing this we had advantages of further orthogonality and low coupling of the whole system. The services sub layer controls the actual functions of the presentation layer. The domain sub layer has the specific description of the individual view types. The business layer works as an agent between the presentation layer and the utilities layer.

- **The Utilities layer** is the backbone of the MaramaSUDDEN system. It comprises different java libraries (e.g. java.swing and java.swt), external java libraries (e.g. Jena), and plug-ins. These libraries and plug-ins provide the foundation of the MaramaSUDDEN modelling environment. They also provide all essentials for the GUIs and functions.
- **The Data layer** consists of two data repositories. These repositories keep the data in different format. One is in a semantic web format; the other is in the MaramaDiagram format which is an XML encoding. The data can be saved and retrieved from both repositories. The main storage repository is the semantic web repositories, since it is independent from any applications, and neutral from any platform. The semantic web repository also can be accessed and communicated by other applications. It is more flexible and is good for knowledge sharing.

Figure 33 shows the architecture of our support tool. Marama provides visual tool designers (1) whose specifications are loaded by an Eclipse IDE instance (2) to instantiate the MaramaSUDDEN tool's models (3) and behaviours (4). The presentation layer comprises a view based visual modelling interface, GUI windows, messages, browsers and etc (5). The business layer defines and describes the specific problem domain/sub domains of the SUDDEN model (6). It also specifies the behaviours of the sub domains. The utilities layer (8) comprises different java libraries (e.g. java.swt), external java libraries (e.g. Jena), and plug-ins. These libraries and plug-ins provide the foundation of the MaramaSUDDEN modelling environment. The data layer consists of two data repositories. One is in a semantic web format (9); the other is in the Marama meta-tool format (7), which is an XML encoding.

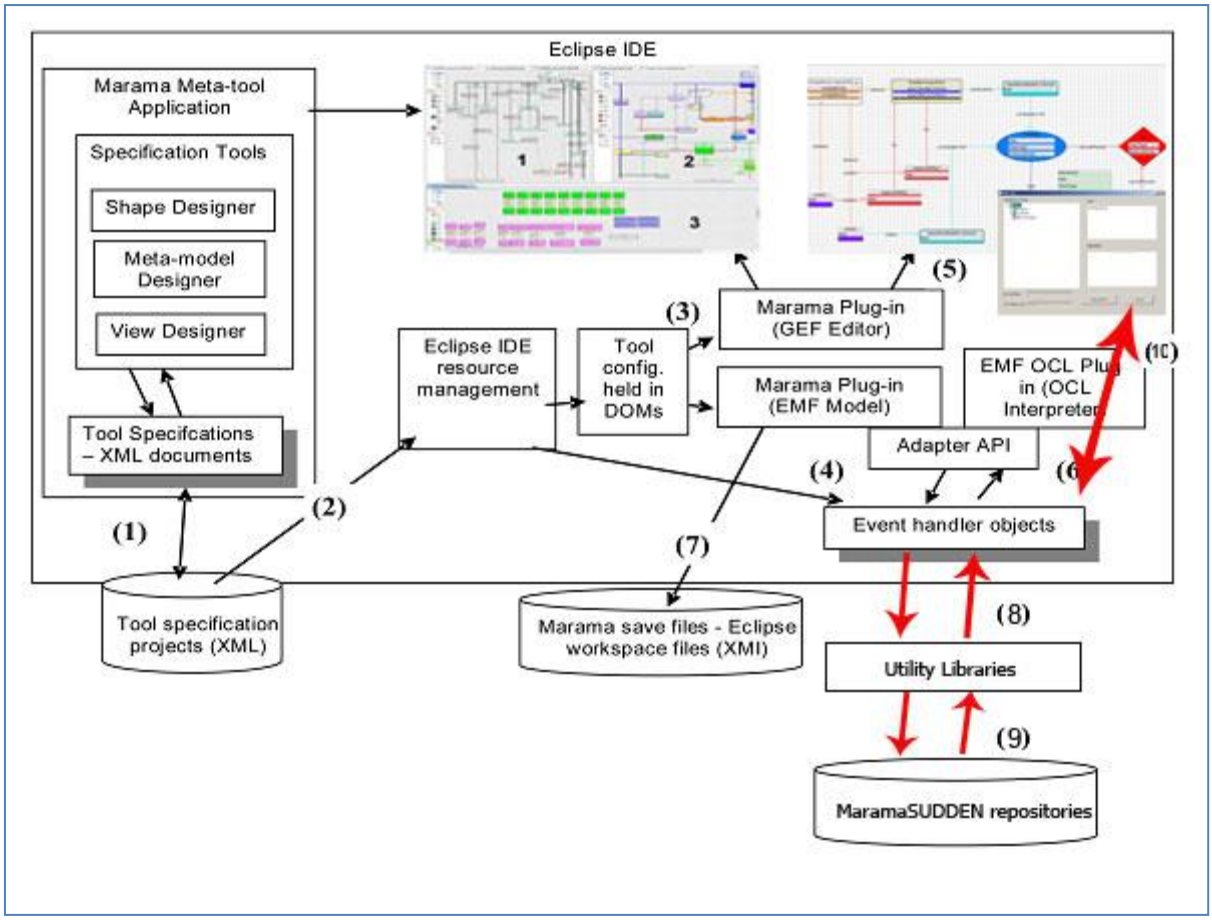

Figure 33.The Architecture of MaramaSUDDEN.

6.3.3 Design view types

Our initial view design took a Representational Epistemology (REEP) approach (Barone & Cheng, 2004), which offers a complex single view (John Hosking, et al., 2008). However, as the SUDDEN concept model was developing, there are several dozens of concepts and relationships need to be visualised. It is overwhelming for users to use all these visual symbols in a limited screen space. To improve the usability of the visual language, we split the original one single complex view into three different view types to allow users to solve different sub domain issues respectively. The trade-offs of using multi-view approach include the increasing level of hidden dependencies, and consequently increasing our concerns for data consistency of instances when they are shared by different views. We feel the advantages of "juxtaposition of views and appropriate visualisations of diagram differences" (John Hosking, et al., 2008) outweigh these potential negatives. Hence, the multi-view approach is appropriate for our tool. Since the Marama platform supports a multi-view

approach, this made our multi-view design much easier. We discuss all three views with example usages are as follows.

6.3.3.1. The Product View

The Product system of the SUDDEN concept model provides different potential decompositions and possible combinations of products and services within supply chain networks. This allows users to query across pre-defined knowledge, e.g. "Simple Part", or create new classes or instances. The design purpose of the Product view type is to allow users to add values (e.g. manufacturing products) for the supply chain networks modelling. Figure 34 shows product decomposition in the Product view type by using the Product DSVL. In Figure 34, a car door can be made from different possible parts. The numbered symbols indicate the key domain concepts used this view type.

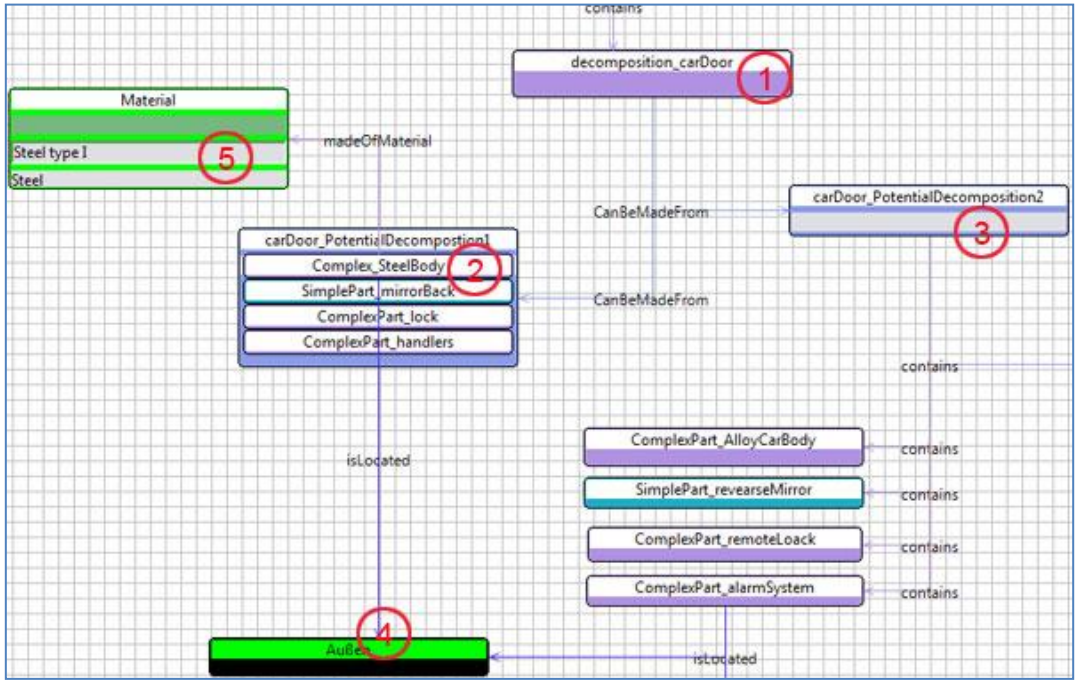

Figure 34. A simple product decomposition example in the Product view.

1. A car door (No.1 shape) is a "Complex Part", which can be made from a range of "Potential Decompositions", e.g. (No.2 and No.3).

2. Each of the "Potential Decompositions" (No.2 and No.3 shapes) includes a set of "Simple Parts" and "Complex Parts".
3. For No.2 "Potential Decomposition", it comprises a "Complex Part" Steel-car- body, a "Complex Part" generic lock, a "Complex Part" generic handler, and a "Simple Part" generic rear-mirror.
4. For No.3 ""Potential Decomposition" , it comprises a "Complex Part" Alloy-car-body, a "Complex Part" remote control lock, a "Complex Part" alarm system, a "Complex Part" generic handler, and a "Simple Part" generic rear-mirror.
5. The "Complex Part" Steel-car- body in No.2 shape is "made-of-Material" "Steel" (No.5 shape - "Material").
6. The "Complex Part" Steel-car- body in No.2 shape is located in "Auben" (No.4 shape- "Location).

6.3.3.2. The Process View

The Process view type allows users to modify and edit processes of a supply chain network based on "Goals" by using the Process DSVL. "Actors" are responsible for decision-makings (Mehandjiev, et al., 2009).

The Process view of the SUDDEN model adopted three basic dependency types from the MIT Process Handbook (Malone, et al., 2003), which are "fit", "flow "and "sharing". This is one of the key concepts of the SUDDEN process model - how the processes can be coordinated with each other, and how the processes can interact with resources (Mehandjiev, et al., 2009). Figure 35 indicates three dependency types from the MIT Process Handbook, which are "sharing", "fit" and "flow". The blue oval shape stands for a generic "Process" and the red Rhomb shape stands for the dependency type.

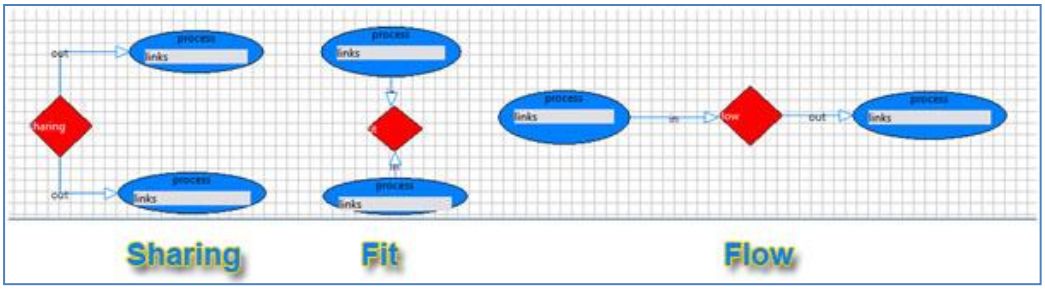

Figure 35. The three basic process dependencies from the MIT process handbook.

Figure 36 indicates a goal-driven process coordination for a supply chain model by using the Process DSVL. The numbered shapes are the major sub notations used in this view type. Since the Process system of the SUDDEN model is goal-driven, we start with a generic "Goal" (No.1 shape).

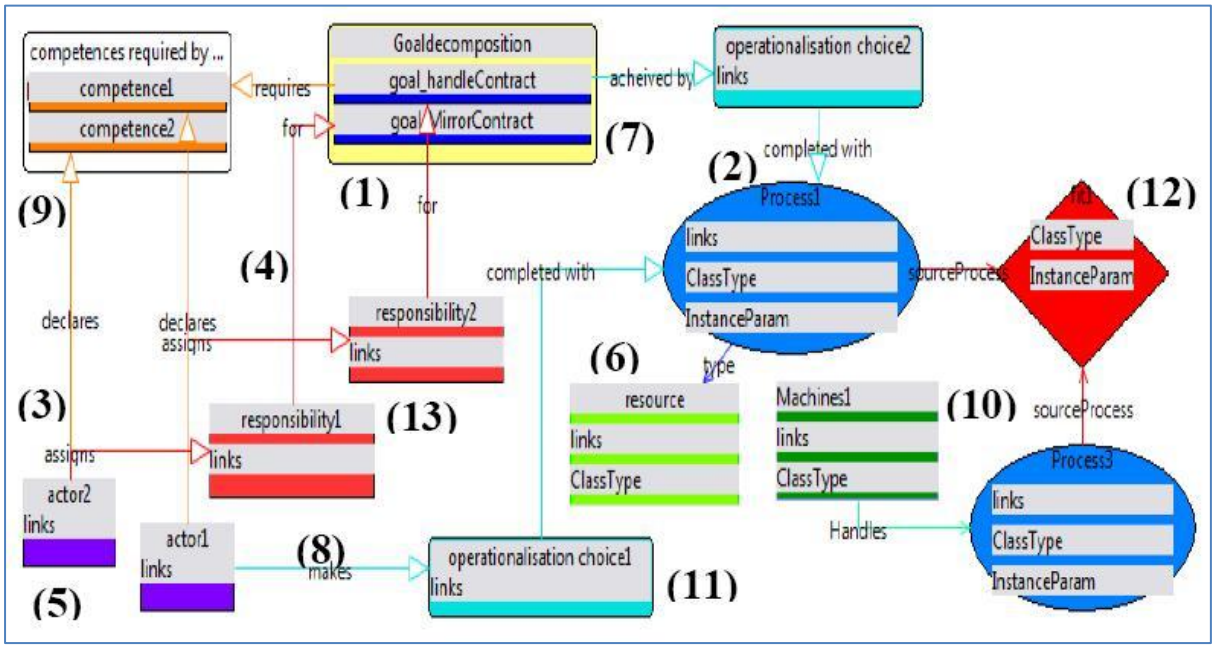

Figure 36. A generic example of goal driven process coordination.

1. In Figure 36, the goal (1) could be a contract to supply car doors within supply chain networks. The decomposition of this "Goal"(No.1 shape) has been represented a set of sub-icons inside of this "Goal", which are the sub-goals, e.g. subcontract of supplying parts for a car door maker.
2. No.5 shape is a "Actor", which could be a contractor or an organisation (Mehandjiev, et al., 2009).
3. No.13 shape stands for "Responsibilities". The "Actor" can be assigned to "Responsibilities" to make decisions about a certain goal. For example, if an Actor obtains a subcontract for supplying car door, who has the responsibilities to make decisions for this subcontract.
4. However, if an Actor (No.5) wants to obtain some "Responsibilities" (No.13) for a certain goal, the Actor must match some "Competencies" (No.9 shape) to qualify for "Actor" selection.

5. Once the Actor assigned "Responsibilities" for a certain goal, the Actor can make decisions, which are "Operational Choices" in the SUDDEN model (No.11 shape), for example, completed the goal by using a certain process, outsourcing, or further decomposition of this contract.
6. In this example of Figure 36, the "Actor" decided to complete the goal with a "Process" (No. 2 Shape).
7. "Processes" also coordinate with other "Processes" and "Resources" (No. 6 shape), for example, some machineries.
8. The SUDDEN model addresses "the reversion of a (sub) process description to a goal statement as discretionary step definition. This provides a certain symmetry in our model which neatly closes the design loop" (Mehandjiev, et al., 2009).

6.3.3.3. The Abstract Supply Network (ASN) View

The Abstract Supply Network view type allows users dynamically to select partners, and to form supply chain teams by using bottom-up approach, which "allows suppliers to form consortia which offer innovative combinations and bundling of activities; whilst '*Select Partners'* allows systematic selection" (Mehandjiev, et al., 2009). Figure 37 indicates a simple ASN forming example.

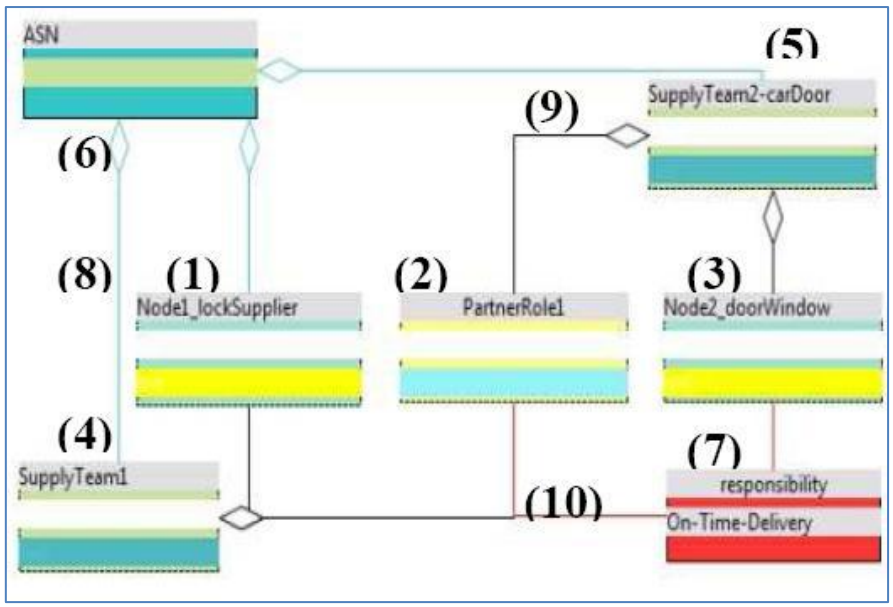

Figure 37. An ASN forming example in the ASN view.

The Abstract Supply Network (ASN) DSVL allows users to dynamically select partners, and form supply chain teams in a bottom-up approach. This lets suppliers form consortia that offer innovative combinations and bundling of activities. An alternative automatic '*SelectPartners'* allows "systematic selection" (Mehandjiev, et al., 2009). In Figure 37 (1) and (3) are "Nodes" of virtual enterprises (Mehandjiev, et al., 2009). A "Node" could be a person or a company. (1) is a car lock supplier; (3) is a car door window maker. To form a supply chain team (4, 5), Nodes must find other suppliers (Partner), (2). In Figure 37, (2) is a steel car door body supplier. (1) and (2) form a partial supply team (4); and (2) and (3) form a partial team (5) to potentially supply car doors. This information can be seen by all SMEs in the supply chain network. Other suppliers (e.g. a car door handle maker) can offer to form a complete supply team to bid for a potential contract. Team1 (5) and Team2 (4) form an ASN (6). (7) is a "Responsibility" class, e.g. On-time-delivery. Both "Nodes" and their "Partners" share "Responsibilities". This is the same "Responsibilities" class that is used in the Process view type.

6.3.4 Interface Design

We used several techniques and methods to improve the usability of our modeling environment: Modularization, Expansion, Simplification, Optimization, and GUIs support. We present a discussion below.

6.3.4.1. Modularization

In order to reduce the screen space waste, modularization technique is used to encapsulate shapes. In other words, a group of similar instanced can be scaled down into one single visual icon. The encapsulation facilitates the scalability of the supply chain modelling. Figure 38 indicates an example of encapsulation in use. In Figure 38, each "Potential Decomposition" instance contains different combinations of parts, which could be shapes or diagrams. In this case, it contains a SimplePart1 and CompexPart66. Both SimplePart1 and CompexPart66 shapes have been encapsulated within the Potential Decomposition shape.

By using encapsulation, a group of instances and diagrams can be modularized into one single symbol to save screen space. By using the nested shape approach, not only is the

number of line crossing reduced, but also screen waste is reduced dramatically. However, the trade-off of using this technique is it increased hidden dependencies.

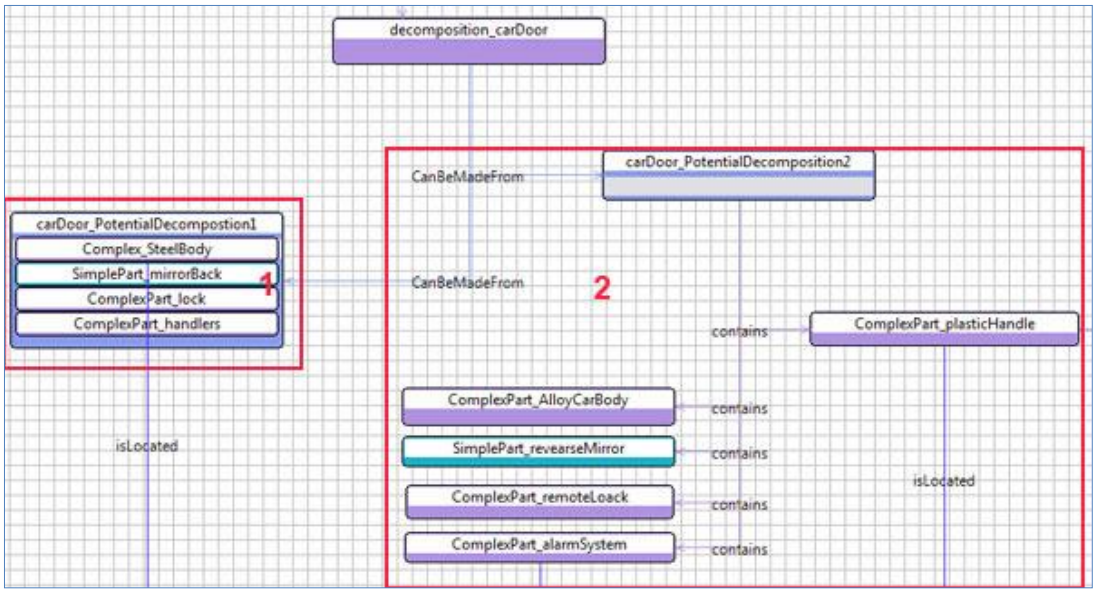

Figure 38. A set of instances (2) have been modulized into one symbol (1).

6.3.4.2. Expansion – Hierarchical Abstraction Support

For supply chain network modeling, the models can become very complex. There could also be many possible alternative options (e.g. potential decomposition of a product) in a dynamically changing supply chain environment. Alternative options are also called "what-if" scenarios. These scenarios provide optional choices for users. For instance, when a shape contains sub-icons, e.g. the most left shape in Figure 39, the sub-shapes could be used to stand for a further decomposition views. We used an expansion technique to achieve this design goal in our modelling environment. Our modelling environment allows our users to open these alternative views by clicking on these encapsulated sub-shapes. The encapsulated diagram will open separately. Figure 39 indicates a hierarchical abstraction of the expansion mechanism.

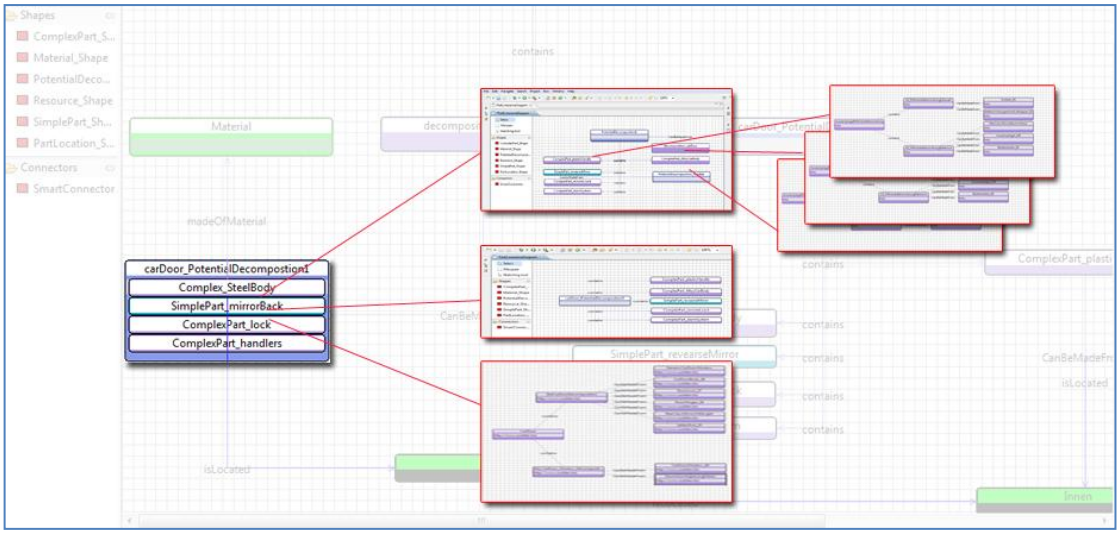

Figure 39. The hierarchical abstraction of the expansion technique.

6.3.4.3. Simplification

Each of MaramaSUDDEN view types has a set of visual metaphors relating to the SUDDEN concept classes and relationships. Due to the limitations of computer screen space, as the number of concept classes increase the palettes of these view types got more crowded, as each of these view types has dozens of relationships and classes. The palettes are too crowded to use easily and we decided to simplify them. We cannot reduce the number of notations for the classes from the palettes. However, we can reduce the number of relationship notations. Our approach is to use one generic relationship type in the palette to represent all of the SUDDEN relationships combined with some artificial intelligence practices to specialize to a specific relationship type. Once the generic visual connector is triggered to connect two visual elements, and if there is a valid relationship between the two classes, the system will create the appropriate relationship instance. Figure 40 shows the palette has been simplified by reducing the number of the connectors.

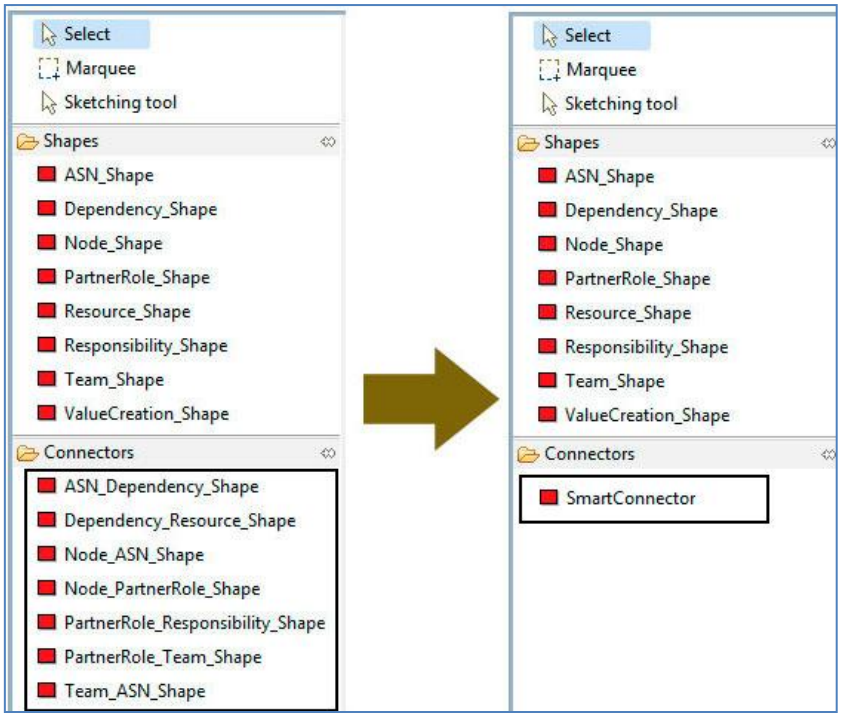

Figure 40. The number of connectors in Palette is reduced to only one.

6.3.4.4. Optimization

Due to the constraints on computer screen space, there is a number of line crossing and screen space waste issues that are harmful for the usability of our tool. To optimize the usage of computer screens, we introduced different line routers to our modeling environment. The users can arrange and layout the diagrams nicely and professionally. Figure 41 shows a rectilinear line router is used in the visual interface design that helps users to organize the layout of the shapes. The connections between the different have been aligned nicely instead of crossing each other.

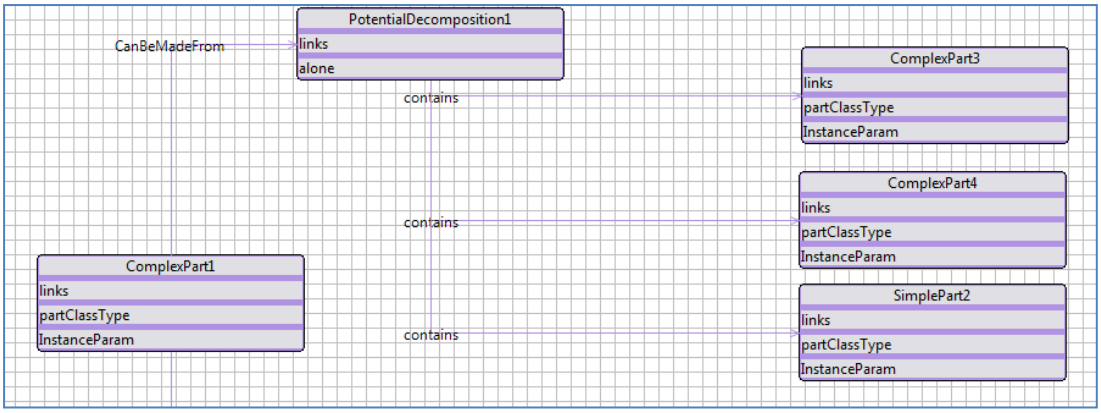

Figure 41. Line routers used to optimize ling crossing and minimize space waste.

6.3.4.5. GUI Windows Support

Apart from the visual modeling interface, we designed a range of GUIs, such as control windows to browse the data repositories. Apart from the main modeling interface, the extra GUI windows make users' modelling work much easier, for example, browsing different parts of an existing model in knowledge repositories. Figure 42 indicates a GUI control is used to select existing classes or instances, or to create a new instance /classes for "Process" shape type. All GUI control windows share a similar style. For Figure 42, the left side of GUI window (a "Class" window) uses a "tree" hierarchy to organize the retrieved names of the existing classes and sub-classes. Users can see and select the existing names from the knowledge repositories for the selected the shape in the modeling interface. The right top window (an "Instance" window) displays a list of names of the existing instances for the selected class in the "Class" window, which also supports "see-and-select". The right bottom window (a "Relationship" window) shows the existing relationships of the selected instance of the "Instance" window.

There are two text boxes at left bottom of Figure 42 to get users 'inputs for new class names and new instance names. If the instance name is empty for a new instance, a unique name will be generated by the tool.

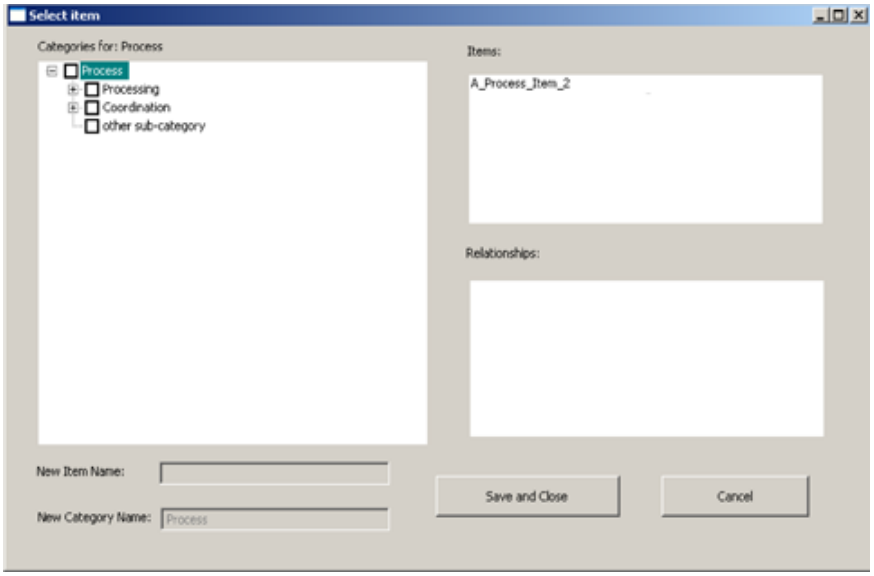

Figure 42. A GUI control window to select existing instance names or create a new one for selected shape type.

6.3.5 Facilities Design

6.3.5.1. Real-time/non-real-time Editing Mechanism Support

To assist different levels of users (e.g. novice to professionals) at editing time, we designed two editing modes, one is real-time and another is non-real-time. These two editing modes allow our users to communicate with the data repositories in either real-time or non-real time. One of the advantages of using this on/off mechanism is that users can switch editing modes at any time. Figure 43 shows the communication between the modeling interface (top) and the database (bottom) can be either synchronized or asynchronous.

In real-time editing mode, users can use the visual modeling environment to manipulate the knowledge repositories in two ways: retrieve pre-defined data from the template repositories into the modeling environment, or users can add, delete, manipulate the models on the computer screen, then changes take effect instantly in the knowledge repositories.

The real-time mode was designed for mission critical and high-performance work. The real-time-editing mode requires that users have rich experience of the visual modeling environment and expert level domain knowledge. It facilitates real time data synchronization and communication between the modeling environment and the template repositories.

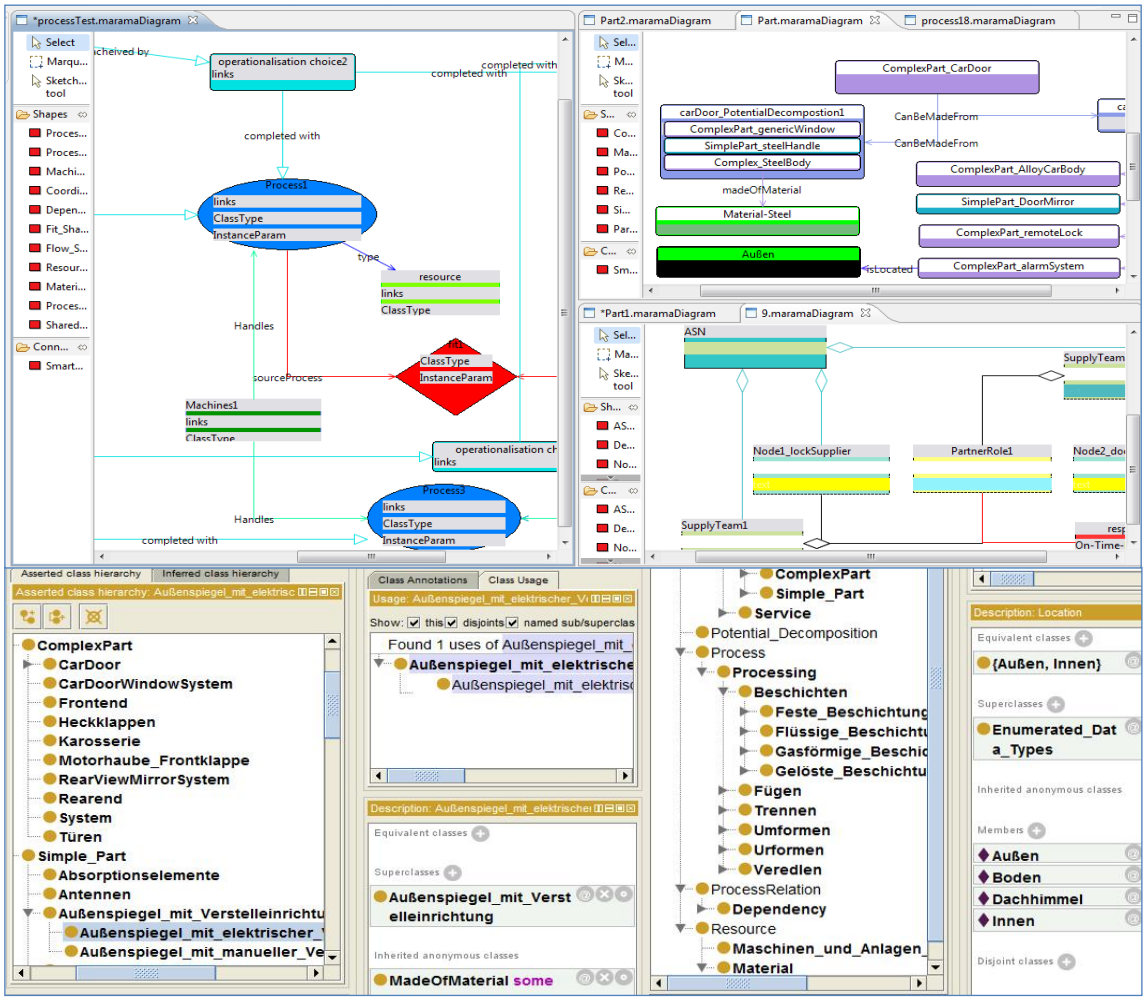

Figure 43. The communication between the interface (top) and the database (bottom) can be synchronized or asynchronous.

The non-real-time editing mode is very useful for users at beginner level, or users needing to build up a model from scratch where lots of errors need to be tolerated. The Non-Real-Time mode allows users to make and correct mistakes at editing time and no instant changes are made to the data repositories. Only the final well-designed model will be saved into the repositories. In non-real-time editing mode, the interface works as a buffer between the user's model and the data repositories, and the data communication is asynchronous. Furthermore, it is more fault tolerant for beginner level users than using real time mode. The users' actions do not occur immediately. For non-real-time editing, once users decide to save changes into the template repositories, the system will save users work into the repository. The system will save all instances first, then the relationships. This is important to keep data consistent. Otherwise, if no valid instances for some relationships in the data repositories, the system could not successfully save some relationships. For example, a sales transaction must involve at least one buyer and one seller. You cannot set up a partnership for one party.

6.3.5.2. Consistency Validation Mechanism

Consistency management is an important issue for modeling tool design. Since our tool supports multi-view, multi-diagram facilities, synchronised and non-synchronised editing modes, all these mechanisms raised our concerns f or inconsistency. For example, normally users only work on a part of a supply chain model and they are not necessarily able to see the rest of the model, which causes heavy pressure on consistency management.

Our tool design was facing different kinds of consistency issues but mainly caused by different times and locations. We classify these issues into four affected areas are as follows,

- The consistency issues are caused by the same instance used in different views or different diagrams.
- Consistency issues are caused by time delay (real-time/non-real-time).
- Consistency issues are caused by starting point of the work (location and time).
- Consistency issue are caused by violating the syntax or semantics (this is less serious, normally caused by users at run time).

These consistency issues have significant influence on the usability and performance of our visual modelling environment, especially deleting, adding and manipulating. Software engineering community has proposed many solutions to facilitate inconsistency management (Chang, 2005). However, there is no single silver bullet to solve all inconsistency issues. Therefore, we used combined mechanisms to validate and maintain the data consistency. As the result, our users can perform design smoothly.

Our consistency validation mechanism includes a locking mechanism and a validation mechanism. The mechanism is designed to handle the consistency issues when end users updating or changing their work.

1. Locking Mechanism

Our locking mechanism is very simple, and works for different conflicted changing and sharing scenarios. If a users' modification impacts any model instance constraints, the object being changed will be locked to prevent potential damage. For example, if an instance has different relationships with number instances in different diagrams, and if a user cannot see all the relationships from the diagrams, if the user wants to delete this instance, the locking mechanism will prevent the instance from deleting due to the constraints of its relationships

in the knowledge repositories. We used a locking mechanism to make sure that shared models/work is consistent. Figure 44 indicates a relationship between D and B in the ontology file is a constraint for locking D in the ontology (left) when a user deletes another copy of D at the modeling interface (right).

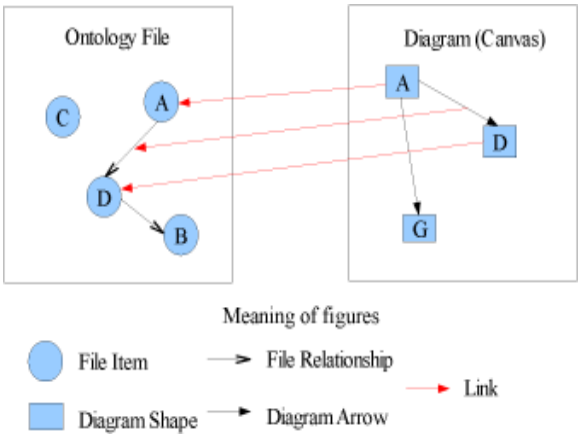

Figure 44. A relationship between two instances (D and B) has a constraint to lock D o(left) from deleting. The drawing is credited to Jesús Héctor from the SUDDEN project.

2. Validation Checking Mechanism

The first part of the validation checking mechanism is to make sure the user created instances are valid against the MaramaSUDDEN meta-models. Our validation system checks any changes made by users against the meta-models. The system checks the syntax for the user's work, which is especially useful for novice. For example, novice users could try to create relationships which are not valid between two instances. If a user created instances are valid by the checking mechanism, the action will take effect. Otherwise, the system sends error messages to users instantly, and reverses the user's action. The validation checking system is to make sure users' works is well-formed against syntax at meta-model level.

The second part of the validation checking is against the knowledge repositories, to make sure that users' work is consistent with the data repositories. The validation checking mechanism makes sure there is no constraint from other diagrams or views to prevent modifying a user's work. (The user 'action is unrestricted). In real time editing mode, since users' actions directly affect the data repositories, users' actions are monitored by the validation system synchronously. However, in non-real-time editing mode, the

synchronization only happens when users' save their work into the data repositories. The consistency of the system could be affected by deletion and manipulation, or adding new relationships. For instance, if a user deleted an instance in the current diagram, but this instance also is used by other relationships in other diagrams. The validation mechanism will check users' action against the data repositories to prevent this instance being removed from the data repositories. Furthermore, the system shows the relevant instances and relationships with related to users' changes. We believe it would be frustrating if users really need to change it something but they could not. Therefore, the users can make the final decision whether to have the change take effect or not. Figure 45 shows the validation checking mechanism is in use. In Figure 45, user's action in view 1 (delete a shared instance by both view1 and view2) is not valid against the ontology database (view 3) by the validation checking mechanism. Combined with the locking mechanism, they form a effective consistency management solution for our support tool against both the meta-tool data and the ontology repositories.

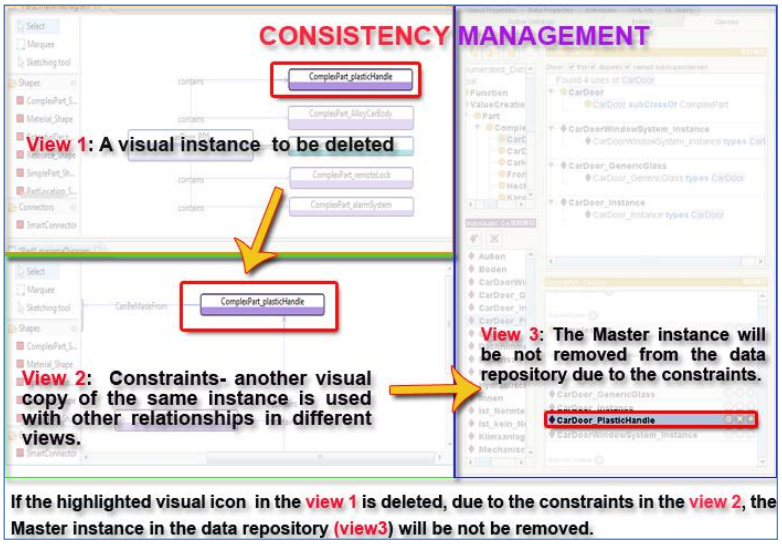

Figure 45. Consistency Management.

6.4. Function Design

In our requirement analysis chapter, we defined the key use cases for our modelling environment. To fulfil the requirements, we developed a set of functions for our MaramaSUDDEN tool. The functional design is based on Model Driven Development approach. We used different sets of analysis and design diagrams to convert the use case models into concrete platform independent models, for instance, activity diagram models. These functions models form the final behaviours of the modelling environment.

Due to limited space, we do not want show all the diagrams here. To indicate how the functions work, we present the "select and set ontology" function design as a sample to show how one of the design models assisted our function design. For our design models, the sequence diagrams help us view interaction analysis between lifelines as a time-ordered sequence of events. Our experience is that they are one of best interaction models for use case realisation. Figure 46 is an example of use case realisation for the function "select and set ontology".

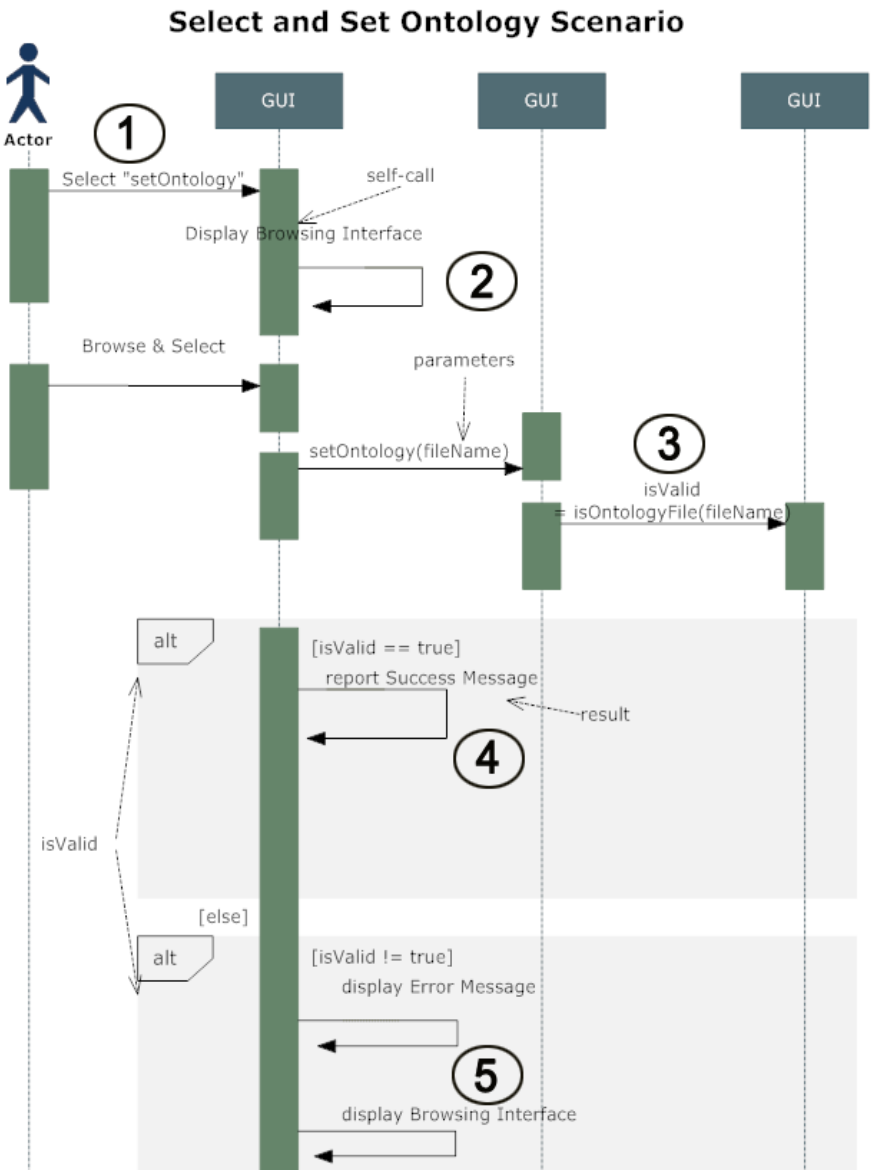

Figure 46. An example of use case realisation for the "Select and Set Ontology" function.

Below is a walkthrough for the function "select and set ontology" as a sample use case realisation.

1. In Figure 46, (1) shows End Users trigger the function, which sends a message to the tool to activate a GUI window
2. (2) shows The GUI window pops up to allow users to browse and select ontology files
3. (3) shows once a user selects a file, the system will get the selected file name and check and validate it against an ontology file. Based on the result of validation, there will be two alternatives
4. (4) shows alternative 1, the selected file is valid.
   4.1 The system sends a message to the user that the selection is successful
5. (5) alternative 2, the selected file is not valid.
   5.1 The GUI sends a message to the user that the selection is not valid
   5.2 The file selection window pops up again for users to select again.

Table 4 shows a set of functions we developed for the MaramaSUDDEN environment by using the same design modeling techniques. The name of the functions should speak for themselves.

Name of the Functions	Function Description
1. Select&Set Ontology	To select a valid data repository, and set up a connection with the visual modelling interface.
2. Select&Set Translation File	The User selects and set a valid translation file.
3. Search Class	The User searches information for a given class name in the database
4. Link Instance	1. To link the visual icon to an existing instance in the data, 2. or to create a new instance for a existing class/subclass, 3. or to create a new instance and create a new class or subclass.
5. Delete Instance	The User deletes an instance from the database.
6. Create Tuple	The User creates a tuple between two valid instances in the database.
7. Remove Tuple	The User removes a tuple from the visual modelling interface and the database.
8. Open Link	The User opens a hyperlink in a Web browser
9. Open Alternative View	The User opens a modularized object in a separate view.
10. Browse Ontology	The User browses information from the

	database.
11. Unify Shape Size	To set all visual notation to the default size.
12. Switch Real Time/Non Real Time	The User switches the editing modes of the visual modelling environment between the real-time and non-real-time modes. In real-time mode, the data communication with the database is synchronised, in non-real-time mode, it is not.
13. Save All	The User saves all his/her work into the database.

Table 4. A set of functions to assist users in the modelling environment.

6.5. Summary

In this chapter we have discussed the design of our MaramaSUDDEN tool. We gave an overview of the design of our tool, described in detail in some areas of the design and some of the design decisions and justification. We put key emphasis on the architecture and functionality of our support tool, which includes some key mechanisms to solve consistency issues. Due to the time constraints, there is still much room to improve, which the strengths and weakness of our tool will be discussed in our evaluation chapter. In short, it provides an excellent proof of concept realization of the SUDDEN model. Next Chapter, we present the implementation of MaramaSUDDEN.

Chapter 7. Support Tool Implementation

7.1. Introduction

In the previous chapters, we discussed the design for our DSVLs and support tool. In this chapter, we present the detailed implementation our tool. Firstly, we present our implementation approach. Secondly, we discuss some of our implementation choices with reasons for the approaches taken. Thirdly, we discuss our experience of implementing the DSVLs using the Marama meta-tools. Lastly, we discuss the experience and lessons learnt from some of the technologies and techniques used for our tool implementation.

7.2. Implementing Model Driven Engineering (MDE) Technologies in supply chain knowledge management

No enterprise can afford to stay still in current fast-changing marketplace. However, businesses do not always have time to sustain their core processes with fast-changing technologies for long term, especially in fast growing supply chain domain (de Castro, Mesa, Herrmann, & Marcos, 2008). Different studies show Model Driven Engineering is one of the best solutions to solve this challenge (de Castro, et al., 2008; Steffen, 2009).

One of the main advantages of using MDE in enterprise applications is that technologies and the core business models can evolve separately. In other words, the core business models can be accessed and shared by different applications at different levels that are using a variety of technologies (Bezivin, Barbero, & Jouault, 2007). For our tool implementation using the MDE approach, we shifted the focus of tool development from writing code to building good models. Below is some experience and lessons gained from using MDE in our tool implementation.

1. Our tool implementation applied the MDE approach and techniques to transform the domain model from a set of business concepts into visual meta-models. By applying MDE techniques to our tool implementation, we focused on creating solid solution

models for a longer term, not just programming the artefact for present requirements. As a result, the solution model of the supply chain domain is totally independent of the technologies or artefacts.

2. By using MDE, we not only created the meta-models but also the visual notational models of the SUDDEN concept domain. The meta-models act as an agent between the visual notation models and the domain models. Using MDE in our tool implementation increased the traceability between the requirements and the models. As requirements change, our users can easily update domain models and trace back requirements changes. Both the concept models and visual notational models are separate, and can be maintained respectively.

3. By applying the MDE approach in our tool, we developed different view types by breaking the solution domain into sub-domains, which focus on different parts of the problem model. The view types include the Product view, the Process view and the Abstract Supply Network view. These views types are powered by the same meta-models, but they are not part of the meta-models. The view types are easily edited and refactored.

4. By using MDE, the data repositories are maintained independently in a form of semantic web ontology. Users' work can be saved into any selected data repositories for sharing and reuse. Users can manage their work (knowledge) in the data repositories via the interface of our tool. In addition, the knowledge kept in data repositories is also independent from any system or platform.

7.3. Implementation Choices

Since our tool development was time constrained we preferred to use efficient and simple solutions in our tool implementation, which are the most efficient options for our users with regard to usability.

Our tool implementation focussed on how best to assist users to model their work. During our tool implementation, we gained considerable value from having real industrial users available

to provide us rapid feedback and opinions for us to make correct implementation decisions, and to adjust decisions as requirements changed.

7.3.1 Implementing functionality by using Java

1. **Java is a default choice for programming -** Marama is a set of Eclipse-based plug-ins developed using Java (J. Grundy, Hosking, Huh, & Na-Liu Li, 2008). Therefore, we always took compatibility with the Eclipse IDE and the Marama tools into consideration for implementation decisions.

   As a consequence of building our tool on top of the Marama framework, Java became the default programming language to develop functionality. For our tool implementation, we used JDK 6 Update 17, which also includes the Java runtime environment. For Eclipse IDE, we used Eclipse EE version 3.5 due to the requirements of the Marama meta-tools.

2. **Experience of using Java for our tool implementation**

   GUI Development - since our tool is a modelling support tool, the usability of its GUIs is a key aspect for our users. We found Java to be very user-friendly for GUI development. It has a range of powerful native and third-party GUI toolkits, which are widely available and free to use. We discuss our implementation choice of GUI **Portability** – our tool also enjoyed portability and multi-platform support from the Java framework.

   Platform independent, Open, free and more - apart from the portability and GUI development friendly features of Java, we also appreciated some other features of Java, which are platform independent, open and free. These features are particularly useful for the deployment and distribution of our tool. In contrast, C# and C++ do not support free and platform independent features.

   A few other features of Java were useful for our implementation, e.g. garbage collector and multi-threading. By using Java, unlike using C++, we do not need to worry about these issues.

7.3.2 Selection of GUI technologies

The visual modelling interface of our tool is implemented by using the Marama tools. We still need to manually implement other GUIs, e.g. interaction windows and messages. As a support tool for visual modelling, the GUIs are the major building blocks for the presentation layer. And the usage of GUI has significant impact on the usability of our tool. As mentioned earlier, there are a wide range of Java GUI technologies and toolkits available to implement our GUI components. We explored several GUI technologies and toolkits, including Java AWT, Java Swing, and IBM' SWT. Since our users are primarily PC users, we mainly considered PC-oriented GUI technologies rather than any others. However, these selected GUI technologies are all compatible with other media, e.g. web or mobile phone. They all can be adopted potentially for our future work.

AWT is a set of GUI toolkit which was developed by Sun (Sun Microsystems, 2002). AWT refers to Abstract Windowing Toolkit. It has fundamental functions to build graphic interface However, AWT does not support many component types which we need in our tool. The Swing toolkit has become an essential component of Java platform since it is a part of JFK. However, Swing has some disadvantages. For example, developers need to be very cautious about the way to design and develop GUIs by using Swing. If not, the applications can be very sluggish. SWT uses peer implementation to build GUI applications like AWT. SWT introduced a group of facilities and services to handle some computer screen issues which is much better than AWT (Feigenbaum, 2006). In terms of the performance, Northover and Wilson state that SWT applications are more effective and efficient than Swing. Since SWT applications use many classes from shared libraries and they only load once (Northover & Wilson, 2004), they use less system resources compared with Swing and AWT. In short, since Marama is a plug-in of Eclipse IDE, and SWT is a default package of Eclipse IDE, we considered not only about the advantages of SWT toolkits in the performance, availability of component types, but also the compatibility with the Marama meta-tools. After comparing above commonly used GUI technologies and libraries, we decided to use SWT as our primary GUI toolkits for our tool implementation.

7.3.3 Choice of Visual GUI editors

Since we chose SWT as our primary GUI technology to develop GUIs rapidly (e.g. code generation and "what you see is what you get"), we decided to use a visual GUI builder for our tool implementation. To ensure the quality of our development, we tested a few popular SWT GUI editors.

We found that Eclipse Visual Editor and Jigloo are the best among all non-commercial Eclipse based GUI editors. The Visual Editor (VE) is an Eclipse plug-in. And it works for both Swing and SWT applications. Jigloo is also an Eclipse plug-in which was developed by IBM. Li and Wohlstadter stated Jigloo is better than Eclipse's Visual editor regarding overall performance (P. Li & Wohlstadter, 2008). Jigloo provides monthly releases, and frequent bug-fixes. We judged that Jigloo is not one of the top level visual GUI editors regarding its features, but it is one of the best among all free editors. We judged that Jigloo has everything we need to implement our GUIs, e.g. tree views. Moreover, it is easy to use, e.g. drag and drop. In addition, the properties of different widgets can be defined and customised in the properties panel with ease. In short, both visual editors are capable to deliver good quality GUIs. But considering long-term support and the compatibility, we finally chose Jigloo as our primary GUI editor.

7.4. Experience of using the Marama tools

In the previous chapter, we compared a range of meta-tools for our tool implementation. We briefly discussed the advantages of using Marama for domain specific visual language development. In the architecture section, we presented our approach on how to build our tool on top of the Marama tools. In this section, we present the details of our visual language implementation using Marama. Figure 47 shows the MaramaSUDDEN visual language is under construction using the Marama meta-tools.

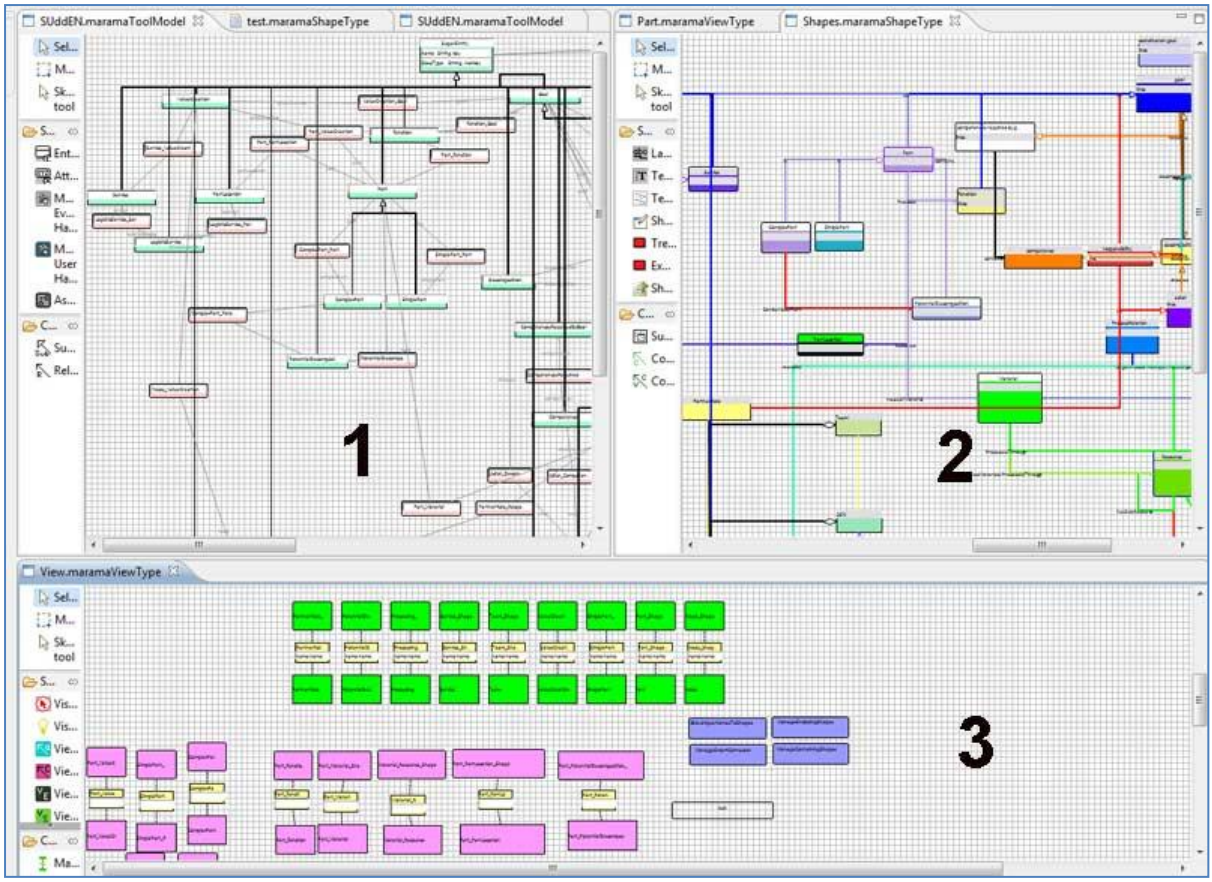

Figure 47. A screenshot of MaramaSUDDEN under construction.

As mentioned earlier, Marama is a set of tools which tackle different development tasks for domain specific visual languages (John Grundy, et al., 2008). The Marama meta-toolset has four major components: a meta-model definer, a shape type designer, a view type designer, and event handler module. Figure 47 shows the key steps to build the MaramaSUDDEN visual languages.

- In Figure 47, (1) shows using the Marama Meta-model Definer to create visual meta-models.
- (2) shows using the Marama Shape Designer to design the visual notational elements for the visual language.
- (3) shows using the Marama View Type Definer to create multiple view types for the tool

We present the details of our DSVLs implementations as follows.

7.4.1 Implementing Meta-Models

To construct the meta-models for our tool, we used the Marama Meta-model Definer, which supports extended entity-relationship models (EER).

The EER models were initially introduced in 1976 for database development (Chen et al 1976). Entities can be used to define concepts or objects. Relationships are used to describe the associations between different concepts or objects. To create an entity in the Marama Meta-model Definer, we just need to drag and drop the entity shape from the palette into the canvas of the Marama Eclipse Editor (all Marama tools are supported by Marama Eclipse Editor).

Furthermore, because all our entities have “name” and a “class type” attributes, we decided to create a super entity with two default attributes (a name and a class type). The other entities are children of this super entity. They inherit the attributes of the super entity.

After converting all SUDDEN concepts to entities, we created the relationships between entities. The Marama Meta-model Definer is capable of handling multi–relationships between two entities. For example, there is more than one relationship type between the two SUDDEN concepts “Part” and “Complex Part”. A “Part” instance can be made from many “Complex Part” instances and a “Complex Part” is a type of “Part”. This scenario requires two relationship types between “Part” and “Complex Part”, which are generation and association. Since the Marama Meta-model Definer supports association and generation type relationships, the multi-relationship modelling is straightforward. Figure 48 shows the meta-models of MaramaSUDDEN under construction using the Marama Meta-model Definer.

Overall the whole process of visual meta-model modelling is straightforward. After that, we needed to create the visual notations of our tool.

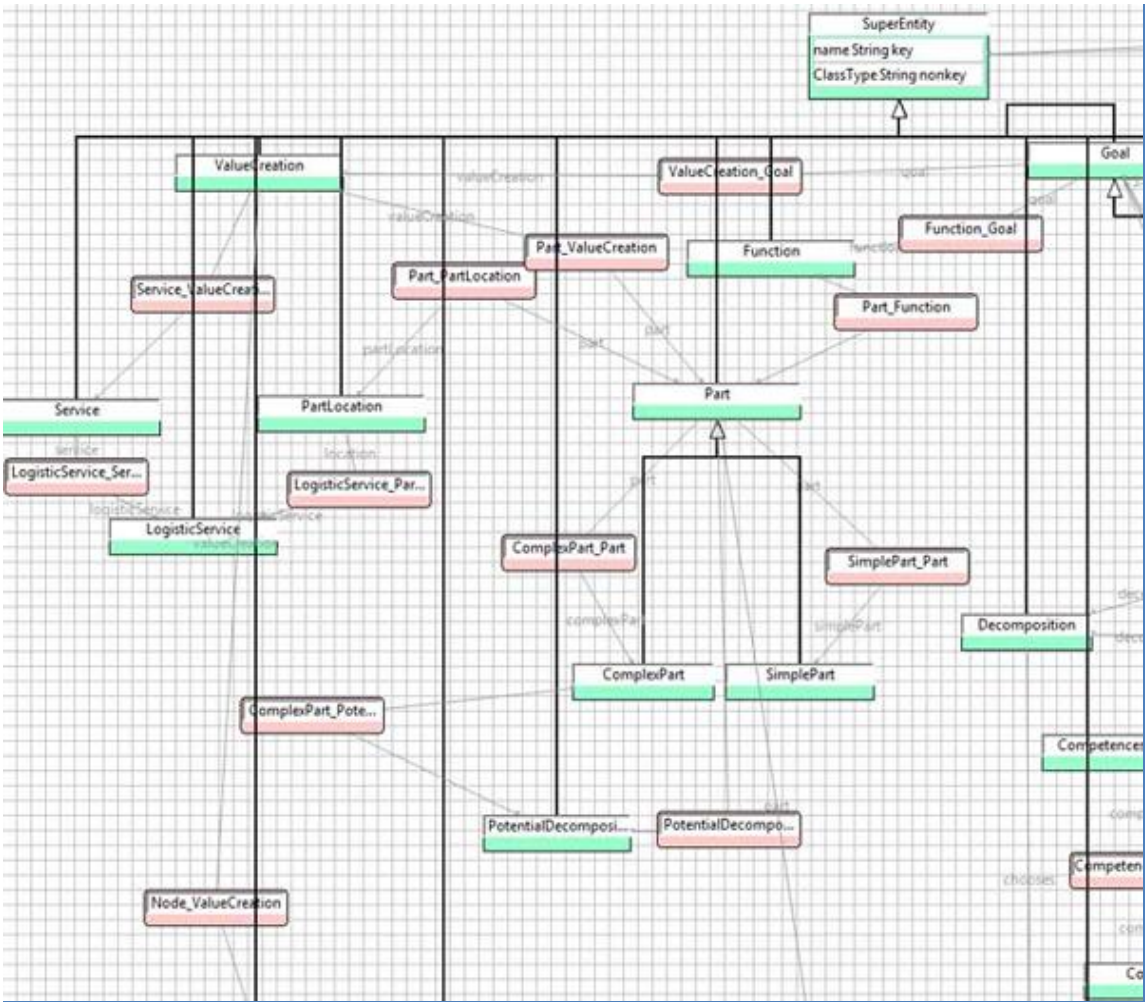

Figure 48. Visualised meta-model modelling by using the Marama Meta-model Definer.

7.4.2 Implementing Visual Notations

To build the visual notations for our DSVLs, we used the Marama Shape Designer to create the visual notations. Each of the SUDDEN concept classes and relationships has been represented by a visual element by using the Marama Shape Designer.

As we mentioned earlier in the meta-model modelling section, between some pairs of SUDDEN concepts, there is more than one relationship type. In the meta-models, we have successfully transformed these into corresponding associations and generations. The Marama Shape Designer allows multiple orientations between two notational elements. Therefore, we created two visual links for each of the directions (A->B; B->A). By using different orientation, we solved most multi-relationship scenarios. However, there were some exceptions. For some multi-relationships cases, semantically both visual links between two notations should be in the same direction (e.g. has, is). For those scenarios, we used extra

popup GUI windows to display possible relationship options to allow users to choose the right one.

Furthermore, as we discussed earlier in the design chapter, we used a set of design elements and templates to design the visual notations for the SUDDEN concepts, e.g. different shape types, colours. All these design elements are supported and available in the Marama Shape Designer, which made our work much easier. Apart from that, the Marama Shape Designer is able to import customised icons and symbols. Figure 49 shows the visual notations of MaramaSUDDEN under construction using the Marama Shape Designer.

In conclusion, we feel that the Marama Shape Designer is very easy to use. The usage and the interface of the Marama Shape Designer are very similar to the Marama Meta-Model Definer. Users can drag and drop different shapes into the canvas of the Marama Eclipse Editor.

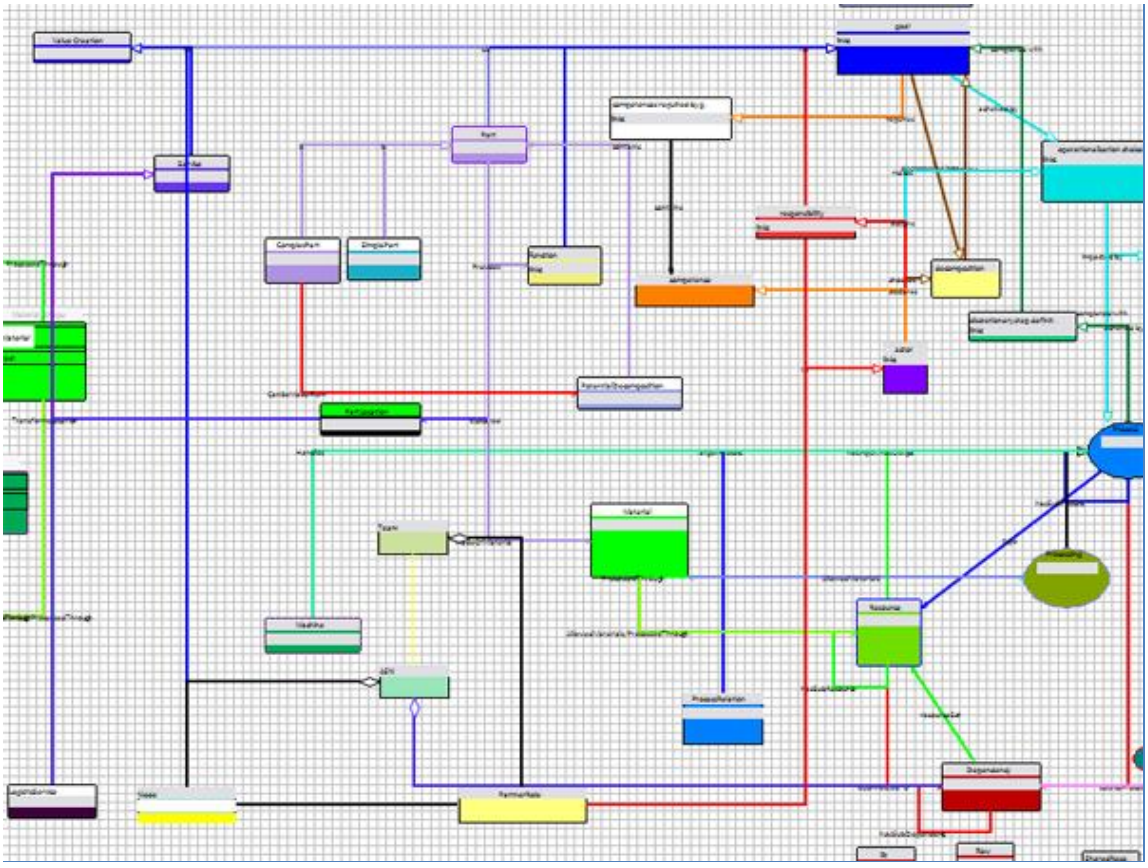

Figure 49. The visual notations of MaramaSUDDEN under construction using the Marama Shape Designer.

7.4.3 Implementing multiple view types

After creating the meta-models and surface notations of our tool, we need to create multiple view types for different sub-systems of the SUDDEN model. The view types can be created by visually mapping the entity of the meta-models to the surface notational elements.

The multi-view technique is an object-oriented software engineering modelling approach. It is a very effective way to split the requirements into different specific responsible environments (Rundensteiner, 1992). The Marama View Type Definer allows users to specify mappings for each of several view types (John Grundy, et al., 2008). We used the Marama View Type Definer wizard to construct four view types: a Product view type, a Process view type, an ASN view type and a big-picture view.

For our users, currently we only need three view types. Figure 50 shows the visualised view type development by using the Marama View Type Definer. However, we feel it is also necessary to construct a big-picture view that potentially facilitates our users to merge different diagrams for different situations (Evans, 2001). To merge diagrams, they could potentially use the existing “extension points”. We explain our idea for potential “extension points” as follows.

The view types are not totally disjoint. Some concepts have been shared across all view types, for example, “resource”. These shared concepts, could be potentially used as “extension points” to merge different diagrams together according to different scenarios. For example, if there are two diagrams, one is a Part view type and another diagram is a Process view type. Both diagrams represent different fractions of a supply chain model. If both diagrams share the same resource, users could potentially merge these two diagrams together.

Figure 50. The MaramaSUDDEN view types are under development.

7.4.4 Overall experiences

We feel there are a few advantages to use The Marama-tools to implement our tool. For example, the Marama meta-tools support MDE modelling. Consequently, the meta-models and the visual notations of our tool are totally separate, which is easy to maintain. By using the Marama toolsets, we could rapidly prototype the visual notations and different view types. Each of notation development iterations took about one or two days. This was due to the constraints of geographically distributed stakeholders and 11-12 hours time zone difference between New Zealand and UK rather than any lack of support from the Marama

toolsets. Also, Marama supports real-time modifications in terms of visual notations, entities, and relationships. Consequently, we can test a range of design elements and ideas instantly. Moreover, by implementing MaramaSUDDEN, we gained experience on integrating a few of plug-ins and third-party libraries with the Marama meta-tools. Due to some technical issues, we had some difficulties to plug one third-party library into the Marama meta-tools at the beginning of the development, but most of the integrating were very straightforward. However, there are some areas that Marama might need to be addressed, e.g. integrating third-party Java libraries support, improvement for real time code editing at run-time instead of editing code at the workspace, and possible default auto-hierarchy layout support for the Marama Shaper definer .

7.5. Implementing multi-threading techniques for our tool

To improve the performance and usability of our tool, we used multi-threading techniques for our tool implementation, especially for GUI windows and concurrent data access. These multi-threading techniques facilitate our tool to operate multiple GUI windows at same time without slowing down the application. The data communications among different GUIs and the data repositories are synchronized by separating the GUI threads and the data access threads.

Multi-threading is a popular programming practice. Multi-threading refers to multiple copies of the same process operating concurrently and independently (Sun Microsystems, n.d.). Multithreading permits all threads of the same process to live within the same environment of a single process (C. T. Wu, 2001). A thread requires less system requires than a process. The Java virtual machine also supports multi-threading applications. Since different threads could access the same data or resource at same time, Java also has a lock mechanism to ensure first in and first serve (except deadlocks). There are many approaches using multi-threading techniques for design, implementation and testing (Seo, Chung, Kim, & Kwon, 2001; Yue-Shan, Lo, Chii-Jet, Shyan-Ming, & Liang, 1998).

To implement multi-threading, there are two approaches. One is to use the Thread class. And another way is to use Runnable. If a class need to call other classes, Runnable is the choice.

Figure 51 is an example of multi-threading used in our tool, which created a Runnable thread from the main visual interface.

```
GraphicalEditorWithPalette editor = (GraphicalEditorWithPalette)
this.getDiagram().getEditor();

final Shell shell = editor.getEditorSite().getShell();

        class AddRunnable implements Runnable {

        public void run() {

        FileDialog dialog = new FileDialog(shell, SWT.OPEN);
        dialog.setFilterNames(new String[] { "OWL Files" });

        dialog.setFilterExtensions(new String[] { "*.owl" });

        String path = dialog.open();

        if (path == null) return;

                SuddenLibrary.setOntologyFile(path);

        }

}

final AddRunnable runnable = new AddRunnable();

editor.getEditorSite().getShell().getDisplay().syncExec(runnable);
```

Figure 51. An example of using multi-threading in MaramaSUDDEN.

When users need to synchronise their interface with the ontology database, they need to select function "select and set ontology". The function will create a separate GUI thread from the main interface editor by executing sycExec().

7.6. Implementing Knowledge Management by using Jena Framework

For our tool, knowledge is kept in semantic web ontologies in OWL format (Martin, et al., 2007). To facilitate our users to manage knowledge via the interface of our tool, we used the Jena Framework to assist with the knowledge management of our tool. The Jena semantic web framework was introduced by HP Lab for managing RDF, RDF and OWLs (Carroll, et al., 2004). We used Jena 2 to support middle layer functions implementation. When the users use the visual notations of the modelling environment to perform tasks, the corresponding Jena APIs will be called accordingly to manage knowledge in the ontology repositories. By using Jena2, our tool can easily parse and manipulate the OWL ontologies. As a semantic web framework, Jena2 also provides rich APIs for reasoning. Our tool can quickly retrieve information required by reasoning in ontology repositories. Figure 52 shows an example of usingJena2 APIs in our tool to retrieve instances of "Resources" from an ontology database.

```
public static Vector<String> getSubResources(String ResourceClass) {
        OntClass cls;
        Vector<String> classes = new Vector<String>();
                if (model != null) {
                if (isResourceAround(ResourceClass)) {
                        cls = model.getOntClass(NS + ResourceClass);
                        classes = getResourceNamesAsVector(cls.listSubClasses(true).toList());
                }
        }
        return classes;     }
```

Figure 52. Jena2 APIs used in MaramaSUDDEN to retrieve sub-resources of a resource.

7.7. Summary

In this chapter, we discussed the approaches taken, our experiences and our lessons learnt from our tool implementation by using The Marama-tools. We presented the implementation decisions we made with some discussion. We also discussed several techniques and technologies used for our tool implementation. Consequently, our tool has improved the performance and usability for both visual supply chain modelling and knowledge management with sufficient visualisation. In the next chapter, we discuss the evaluation of our tool.

Chapter 8. Evaluation

8.1. Introduction

In this chapter, we present an evaluation of our DSVLs and its support tool. The purpose of this evaluation is to demonstrate the utility of our languages and tool and to support its further improvement. The results can assist us to enhance the strengths of our visual notations and tool, and minimise the weaknesses. We split this chapter into two sections. Firstly, we use Green's Cognitive Dimensions (CDs) framework to conduct a usability analysis for our DSVLs and support tool. We present an discussion and analysis against each of CDs' dimensions, which support self-evaluation (Green & Petre, 1996). Secondly, we present the results and a discussion of an informal user test we carried out, and an industrial user evaluation by was conducted the SUDDEN project team at end of this project.

8.2. Cognitive Dimensions Based Visual Language Evolution

Green et al. developed the Cognitive Dimensions framework for discussing information artefact design (Green & Petre, 1996). Green's CDs framework uses a set of commonly agreed terms and definitions to express and discuss information artefacts. The CDs framework has been used for visual language analysis and evaluation for many years (A. F. Blackwell, 2006). For the development of our DSVLs and support tool, we also used the CDs framework as one of the design frameworks for our VL design exercises. Below is a list of selected dimensions from the CDs framework we used in our evaluation, which are described in the context of domain specific visual languages.

- Closeness of Mapping – how well can an artefact express the domain concepts and behaviours?
- Consistency – how well can different parts of an interface fit together?
- Diffuseness/Terseness – how easy is it to express a problem?
- Progressive Evaluation – can you run and test your system before it is totally complete?
- Viscosity – how easy/difficult is it to change or modify?
- Visibility – are all related notations readily available, and if so how transparent are they?

- Hard Mental Operations – activities involve users' intensive brain storming for analysis.
- Secondary notation – can supplemental/informal representations be used to convey extra information or express additional meanings.

8.2.1 Consistency

For VL design, this dimension refers the consistency of the visual interface. If you have learned some parts of the interface, you can understand other parts easily.

For our DSVLs and tool development, we were facing consistency issues against a complex domain model which comprises several dozens of concepts and relationships. Inconsistency of the interface can damage the usability of our VLs. The visual semantics and visual syntax need to be consistent in a single view and consistent over multiple views. We present a discussion for these concerns below.

We took a hybrid approach (a combination of Physics of Notations and CDs) to design our visual notations, in order to make them consistent with the domain model in terms of semantic meanings. The visual interface design extracted the core meanings from the domain model. Then, we translated them into the visual notations using a pre-defined paradigm. We also used a continuous evaluation approach to improve them (John Hosking, et al., 2008). We have been concerned about the consistency among our visual notations in different view types. For our visual notations, the concepts and relationships from the same origin share the same colour themes and shape types, which give our users a feeling of consistency. In addition, strong variations were used to provide distinction between our visual notations. Using these techniques the visual symbols of MaramaSUDDEN are capable being distinguished from each other in terms of syntax and semantics. In addition, our users indicated that the DSVLs can express the concept model very well, and delivers a certain level of consistency feeling for our users.

However, it is hard to measure the users' cognitive understanding for our visual notation design paradigm since people are subjective for the selection of colour theme as a primary notation. But the bottom line is that everyone agrees that the visual notations are cognitively and perceptually understandable, and once you have learned the design paradigms, you can understand rest parts of our VLs intuitive. We feel our visual notations have at least achieved the bottom line (e.g. blue for "value creation", red for "alert").

In summary, for the consistency of our VLs and modelling tool, we believe the most challenging task was to keep visual syntax consistent for a large number of concepts with optimized design. The visual notation paradigm is alive and is capable for further development against requirements change.

8.2.2 Hidden Dependencies

Hidden Dependencies refer to the transparency of the relationships and dependencies among different parts of an artefact. Hidden dependencies "make both understanding and maintenance hard" (Zhifeng, 2001). Zhifeng states (2001) "a hidden dependency is a relationship between two seemingly independent components, and it is caused by the data flow in a third software component". Serious hidden dependencies even prevent further development and maintenance of software artefacts. The hidden dependencies could cause deadly problems, and reduce the life span of an artefact.

Due to the limitation of computer screen space and the complexity of the supply chain model, we introduced a few techniques and mechanisms to gain some screen space for our users, for instance, abstraction and modularization. However, each of these techniques solved some issues, but they also created some hidden dependencies. We present a discussion for each type of these hidden dependencies below.

- Type I hidden dependencies were created by increasing the scalability of the modelling interface. To modularize different views to gain some screen space, we used some abstraction techniques. As a result, some relationships became hidden from opened views. For example, a whole diagram can be abstracted down into one single visual icon, or just a component of a visual icon, while all related connectors of this diagram became hidden. As a result of using these techniques, our tool gains scalability for the modelling interface , but these techniques created some hidden dependencies (John Hosking, et al., 2008).
- Type II hidden dependencies were created by using a multi-view approach. To gain scalability of the modelling interface, we split one simple complex view into multiple views, which gave users more screen space for visual modelling. The multi-view approach modularized the sub domains, but created some hidden dependencies over different views. Some concepts are used in different view types, e.g. "Resources" is used

in three view types. However, the view types do not share all relationships of "Resources", which creates some hidden dependencies.

- Type III hidden dependencies were created by the Non-Real-Time editing mode. To facilitate different levels of users, our tool provides two editing modes, a real-time one and a non-real-time one. In non-real-time, since data communication is not synchronised, our users only can see the dependencies in opened diagrams, and they cannot directly see some dependencies, which actually exist in the data repositories. For example, IA1 is an instance which has two different relationships (R1, R2) in the ontology repository; in the visual modelling interface, an opened diagram only displays A1 and R1, but not R2. When a user wants to delete A1 in the modelling interface, the user would not know there is another relationship of R2 of A1 in the repository.
- Our tool used a number of techniques and mechanisms to mitigate hidden dependencies. For Type I and Type II hidden dependencies, to overcome these problems, our tool uses a locking mechanism to prevent unwanted modifications and changes against constraints. The locking mechanism is used to minimise the side effect of the scalability and multi-view techniques. If there is any constraint on users for modifications, e.g. dependencies, the locking mechanism will lock up modification, and wait for a user's decision. Moreover, our users can use provided GUIs based functions to check all related information, e.g. dependencies. To overcome Type III hidden dependencies, our modelling tool uses a consistency validation checking mechanism to check users' work against the data repositories when users save their works into the databases. We still need to improve this area in our future work, e.g. how to mitigate the impact of the hidden dependency in synchronized editing mode.

To summarise, the hidden dependency issues were difficult to tackle. For our tool, all hidden dependencies were created by introducing techniques and mechanisms for managing scalability and enhancing usability. As in previous work in this area, we feel the benefits gained from the scalability and usability advantages overweigh the hidden dependencies created (John Hosking, et al., 2008).

8.2.3 Error proneness

Error proneness refers to whether a system causes users to make mistakes, and how easy they can recover from their mistakes. Users can make different kinds of mistakes, e.g. a typo. Normally, visual languages and visual tools are more error tolerant than text-based tools, since visual languages provide structured editing, graphic knowledge and data presentations in more readily understandable forms. Users can perform tasks in more natural ways. For our DSVLs and support tool, we outline a few types of mistakes that our users tend to make.

- **Mistakes from Lack of Domain Knowledge** (not really error proneness from classic CD point of view) - since MaramaSUDDEN is a suite of domain specific visual languages for supply chain network modelling, to perform modelling tasks, it requires that users have a certain level of domain knowledge and modelling experience. For example, supply chain domain experts find the visual notations easy to understand, and quickly learn how to use them to perform modelling tasks. However, novice users do not. This lack of domain knowledge can lead to many mistakes and errors. These mistakes are not directly caused by our DSVLs and modelling environment. The best way to help this users group is to improve their domain knowledge. Therefore, we prepared video and text based tutorials, and user guides to introduce the domain knowledge and demonstrate how our DSVLs and support tool works, and how to use the tool to perform some basic tasks.
- **Mistakes by Confusion or Misunderstanding** - misunderstanding and confusion of the visual notations can lead to making mistakes. We believe simple and intuitive visual notations can reduce the level of confusion. To avoid users' misunderstanding for our visual notations, we used a number of variation techniques to make the visual symbols distinguishable. For example, we used strong colour contrast and symbol shape choice for each of the visual symbols.
- **Careless Mistakes -** people can make careless mistakes, which no human can really avoid. For those careless mistakes, we tackled them in three ways, which are conversions, alternative flows, and a feedback system. We used conversions for both the visual notations and the functions of our tool, For example, we used vertical alignment for the nested icons in modularized shapes in our visual notations (Hosking, et al., 2008), and we used see-and-point for functions. Using conversions in our VLs increase the consistency and proximity for users, which reduced error proneness to careless mistakes. Furthermore, for the rest of unpredictable and unexpected mistakes, our tool attacks them via a feedback system. The feedback system is capable to catch all unpredictable mistakes, and

provide instant warnings. The instant feedback includes error messages, notes and warnings. Our feedback system checks users' mistakes against the meta-models and the data repositories. Moreover, users can learn from their mistakes via the message and information provided the feedback system.

In conclusion, since our VLs and support tool does not contain any significant misleading or confusing factors, our users do not have any serious error proneness issues when using our DSVLs and support tool and modelling tool. In fact, our tool helps users to avoid mistakes compared to if they were building SUDDEN models without such tool support. We feel that error proneness of our tool has been reduced dramatically.

8.2.4 Closeness of Mapping

Closeness of Mapping means the level of nearness between an artefact and the problem domain in the context of visual languages. Since the concepts of the domain model are rather abstract, it is hard to say what the level of closeness of mapping is for our VLs. Our DSVLs were mapped to the domain model in two levels. First was the mapping between the meta-models of our visual languages and the domain model. Second was the mapping between the visual notations of our DSVLs, and concepts of the domain model.

- **Meta-Models Mapping** – the SUDDEN model comprises a number of sub-systems to address different aspects of concerns and issues, e.g. coordination of business processes. To map them properly, the meta-models of MaramaSUDDEN cover all these sub-systems, and provide abstract syntax descriptions to support different modelling scenarios. Each of the domain concepts has been mapped to an entity class in the meta-models of our VLs. We feel the meta-models of MaramaSUDDEN have met all the basic requirements as high-level models to describe the syntax and semantics of the domain model. There are no-known mapping issues at this level. Since the Marama Meta-modeller does not supply N-nary relationships, we converted these relationships into many binary ones. However, the mapping process was very smooth. Furthermore, the evolution of the MaramaSUDDEN meta-models and the concept model can be independent. Moreover, for the purpose of model comparison and model differencing, the meta-models of MaramaSUDDEN can be used to trace the changes of the domain model.

- **Visual Interface Mapping** – the closeness of mapping between the visual notations and the domain model means that the visual metaphors of the SUDDEN concepts need to be easy to understand semantically by the domain experts. Moreover, visual notations are capable of communicating each other effectively based on the Physics of Notations framework (Moody, 2009a). In other words, the mapping between the visual interface and the domain concepts need to be meaningful. To map the domain concepts to our visual notations rapidly, as discussed earlier, we used a visual notation paradigm with a group of design elements to design visual icons. We carefully mapped each individual concept of the domain model with our visual paradigm generated visual icons perceptually and cognitively. For example, we carefully mapped colour themes as secondary notations to the core meanings of the concepts. The colour themes of the visual notations ensured the consistency among the visual syntax of the visual symbols. In addition, strong variation techniques provide clear distinction for our users. For example, we used dotted lines for boundaries of the "ASN team" icon, since the dotted lines provide a feeling of "dynamic".

In conclusion, we feel these visual notations are easy to understand and easy to remember. These specially-designed visual metaphors are capable of facilitating our users to perform different modelling tasks effectively. Moreover, to improve the effectiveness of these mappings, we got the stakeholders actively involved with the development and evaluation process. With the active involvement of our users, we improved our DSVLs constantly. The mapping has reached to a reasonably advanced level. We feel most of these visual notations can express the meanings of the domain concepts very well.

8.2.5 Diffuseness/Terseness

Diffuseness means how easy to express a complete meaning using an artefact. The domain model covers many concepts and relationships. Different sub-systems of the domain model focus on different aspects of supply chain modelling.

- **Multi-view approach improves the terseness and the scalability** - To maximise the capacities of these sub-system and modularize the sub domains, our tool took a multi-view approach. Each of view types reflects one of the sub-domains. Each of these view types aims to help users to solve a particular aspect of supply chain issues. We feel the multi-view approach increased the terseness and the scalability of our DSVLs by

modularization of the sub domains. The multi-view approach is a good sample of the long tail theory (20/80 rules) (Andres, 2007). Statistically, our users only need to use average 33% of the total visual icons to express a part of model in one view. In other words, the multi-view approach helps our users to eliminate 100% unnecessary icons, which are average 66% of the total icons in one single view. In addition, the multi-view approach minimised the "icon searching" time. In contrast, without the multi-view approach, theoretically users need to use three times effort and time to search/index visual icons. Therefore, we feel the interface of our tool is terse.

- **Simple and Intuitive Design improves diffusion of our VLs** - due to the complexity of the supply chain modelling and the limited computer screen space, during our design process, we put key emphasis on the rapid visual notation prototyping paradigms and scientific based design rationales (Moody, 2009a).

To assist users to understand and remember the visual icons, apart from using our visual paradigm to design the visual icons, each of the visual notations has two texted-based attributes, which identify the name and the class type of the visual icons. Since our VLs were specially designed to facilitate supply chain modelling, the target users are domain experts and consultants. We expect our users are familiar with the meanings of these phrases, which indicate the nature of the domain concepts. Before they become familiar with the visual notations, they can remember these notations by reading the text first.

8.2.6 Hard Mental Operations

Hard Mental Operations refer to logical brainstorming activities that involve lots of analysis and design. Hard mental operations tend towards being for logical analysis rather than any creation. For our users, how to coordinate different processes is a good example of hard mental operations. In the context of supply chain modelling, the hardest mental operations refer to optimisation and simplification of modelling. For our DSVLs and support tool, it does not require any significant effort to learn and to use. Therefore, our DSVLs and support tool itself does not require any critical hard mental operations.

8.2.7 Premature Commitment

Premature commitment refers to users needing to make a number of plan or design decisions before an artefact is mature. For our visual modelling environment, our users only need to make decisions about the layout and the structure of the visual models. Except from that, there is no obvious premature commitment for our users to perform modelling tasks in our tool. Below are two types of premature commitments that our users need to make.

- **Presentation of Models** – To complete modelling tasks, users need to make decisions on a few matters due to the complexity of the supply chain modelling, e.g. the layout of their works. It is not necessary but it would be good for users to plan their work first before they actually perform any tasks. For presentation of users' work, our tool provides a range of line-routers to reduce the number of line crossings. In addition, the size of all visual icons are unified by default (can be customised by users), which makes it easier for our users to organize these icons in their work. By using these techniques, the users' work is more readable. Users only need to decide how to organize the visual icons in their work.
- **Modelling Decisions** – when our users perform modelling tasks by using our tool for complex supply networks, they need to make decisions about their work, e.g. select a partner, or decompose a product. To facilitate our users in making decisions for their work, our tool provides them capacities to reference existing knowledge from the knowledge repositories. For example, when a user wants to create a new model, the user can reason and retrieve existing relevant models from the database, e.g. the SCOR model, and tailor the existing knowledge to meet their design requirements. With support from our knowledge repositories, our users can learn from existing knowledge and save time spent making wrong decisions. In addition, our users are able to make quicker and better decisions based on the existing knowledge. Users' work also can be processed and saved into our knowledge management system for future use. Users' processed domain specific knowledge can be shared to other supply chain practitioners within virtual enterprises environment (Mehandjiev, et al., 2009) for future decision support.

Overall, our tool does not require any critical premature commitment except the decisions on the layouts and structures of their work. All critical premature commitments relate to the nature of the supply chain modelling.

8.2.8 Role-Expressiveness

Role expressiveness refers to if the visual notations can "speak for themselves" and if the VL can express a complete "visual sentence". The bottom line for any visual language is that all domain concepts have been translated into visual vocabularies, which can form a basic visual sentence. For the best VLs, the concepts of the model have been interpreted into visual forms naturally, and all users find the visual notation intuitive.

For our visual languages, each of the domain concepts is a role in our supply chain modelling tool. A range of visual icons were used to express the meanings of these domain concepts. Moreover, each of the visual icons takes a simple responsibility within the visual interface to help avoid confusion or misunderstanding.

Since the domain model is rather abstract, all visual icons were designed from scratch. Although all visual icons are new, in general our users feel that the visual notations express the domain concepts (roles) clearly and logically. Each of these visual icons was easy to understand and easy to be identified. In addition, with support of these visual notations, our users are capable to create and manipulate their work effectively. Therefore, between the bottom line and the best VLs, we feel our DSVLs are in the middle of this Role-Expressiveness scale.

8.2.9 Secondary Notation

Secondary notation means supplement/alternative visual forms to convey information, and express meaning informally. The bottom line is that a secondary notation is used to provide the likes of comments for the domain experts, but not for everyone.

In our initial design, we used colours as secondary notations. When we adopted our visual notation design paradigm approach, colours with texture have been used for dual coding as primary notations. In the final version of our VLs, we use both hierarchical layout and navigational line routing as secondary notations to help users to understand. For the best cases, the secondary notation can express semantic meanings, and the visual notations can communicate each other by using secondary notation.

- **Using hierarchical layout as a secondary notation** – for our VLs, due to the dynamics of supply chain networks, fully automated layout algorithms are not well suited

(Reinhard, Seybold, Meier, Glinz, & Merlo-Schett, 2006). The computer simulated models need to reflect different properties of underlining models in real world, e.g. locations. The hierarchical layout as a secondary notation includes information e.g. positioning, size, users defined properties (Reinhard, et al., 2006).

- **Using navigational line routing as a secondary notation –** our DSVLs also uses navigational line routing as a secondary notation. We believe that proper line routing (e.g. direct, Manhattan) can create "semantic navigation" for our visual notations. In addition, it is possible for different line routings to express meanings and convey different information. We feel the line routing techniques as secondary notations are very useful, especially when auto layout algorithm does not work in the problem domain. .

In summary, it appears not to take much time and effort for our users to understand and remember our visual notations. The secondary notations effectively facilitate our users to learn our VLs and express additional informal semantic meaning.

8.2.10 Viscosity

Viscosity refers to how difficult/how easy it is to change a model in the system. Normally, when increasing the level of abstraction of visual notations, the level of the viscosity will be decreased (Green & Petre, 1996), and vise-versa .

The complexity and dynamics of supply chain modelling makes that our VLs have high viscosity. To mitigate this, we introduced some techniques, e.g. auto layout for enclosed icons of the encapsulated object (John Hosking, et al., 2008). For example, different combinations of simple parts and complex parts can be encapsulated into one single visual icon. All included sub-icons will move and resize with the "shell". The trade-off is by easing the viscosity; but the abstraction of the VL is increased.

8.2.11 Visibility and Juxtaposability

Visibility refers to if everything needed by a user can be viewed at once. Juxtaposability refers to if different parts of the notations can be compared side-by-side.

Since we took the multi-view approach, domain specific concepts have been scattered into three task oriented views. Therefore, when users use one single view type, they do not see some concepts in other two view types. The reason of using the multi-view approach is to provide proper modelling space due to the nature of the supply chain networks. The trade-off is that it lost some visibility compared with the single view approach. However, since different views and diagrams can be viewed and used side-by-side, it gained some juxposablity to mitigate the loss of visibility by using the multi-view approach.

In summary, we feel the visibility of tool is above moderate, and the juxtaposability of our VLs is good.

8.3. Progressive Evaluation

Progressive Evaluation indicates whether a user can run the system as it is being developed. In our case, with the support of The Marama-tools, our VLs support model checking during their development. For example, if we have created a visual icon and an entity for a certain domain class and mapped with the icon with the entity, then we are able to use this domain concept. As more domain concepts are visualised, we could prototype some models by using these visualised domain classes before the VL specification is totally mature. Since our VLs have three different view types, each of the view types are task specific, they can be used together or individually which depends of the specific tasks. Once one view type is build up, it can be used for some modelling works without another two view types. Hence our VLs can support some model checking based on one single view during its development. Moreover, we also can make adjustments and improvement as requirements change.

8.3.1 Informal User Testing and User Evaluation

To estimate the usability of our domain specific visual languages and tool, we conducted a usability study. It comprised an informal user test which was carried out in Nov 2009 in Auckland, New Zealand, and an industrial SME user evaluation which was conducted by the SUDDEN project team in November 2009 in Steyr, Austria. For the informal user test in Auckland, the participants are visual language researchers and academics. They were asked to fill up a questionnaire after the test. For this user test, we wanted to monitor users'

behaviours and practices during they perform the pre-designed modelling tasks. In addition, the user test was expected to help us to measure the robustness of our tool. Based on the after-test survey, we gained valuable feedback from the participants about a range of usability issues in terms of visual language design.

For the industrial user evaluation, a walk-through demo was provided for four industrial end users. The SUDDEN team was trying to obtain comments and feedbacks from these industrial users during the demo tour. The discussion and comments during the industrial evaluation was videotaped for further usability analysis. For the industrial SME end user evaluations (held in Steyr, Austria) there were a few points that the SUDDEN team wanted to find out from the industrial SME users. Importantly, they wanted to know if MaramaSUDDEN can fit the requirements of different target end users. More importantly, the SUDDEN team wanted to obtain feedback about potential evolution for MaramaSUDDEN. It was difficult for the SUDDEN team to recruit the right industrial participants since the tool walk-through demo requires certain level background knowledge and technical skills. It also takes a large among time for these SME users. Furthermore, another purpose of this industrial user evaluation is to help the SUDDEN team to identify the scope of the target users and identify requirements for further development. The feedback from these industrial SME users also provided us valuable commercial grade insights for our DSVLs and support tool.

8.3.2 Preparation and Plan for the Informal User Test

The informal user test (Auckland, New Zealand) included a tutorial and a set of pre-designed modelling tasks for some basic modelling scenarios.

Firstly, we prepared the tutorial to explain our visual languages and the supporting modelling environment, and related concepts as well, for example, how we mapped the concept model to DSVLs and how we developed their support tool. The purpose of using a standard tutorial was to allow all test participants to gain the same background information. The tutorial consists of a short movie, a detailed user guide and a personal demo, and a practice session. The flash movie and the user manual were used to illustrate how to use our tool to implement some simple modelling tasks. The tutorial was designed to assist the participants to understand our DSVLs and support tool. In the personal demo and practice sessions, the

participants are allowed to ask questions and get hands on us the tool. We gave the participants some time to get familiar with the modelling environment. During the practice session, the participants did some simple modelling exercises on pre-prepared modelling templates.

Secondly, we recruited our participants from academic backgrounds, who have advanced knowledge and experience in visual language design, but a lack of domain knowledge. Therefore, we provided them with more help on the domain knowledge.

Thirdly, we prepared a set of basic modelling scenarios, which involved using a sequence of functions to complete tasks.

8.3.3 Questionnaire Preparation for the informal user test

Once the participants completed the user tests, they needed to fill in the questionnaire. Our questionnaire was designed to obtain general feedback about the usability of our visual languages and the support tool. Since it was a small informal questionnaire-based survey, we did not have a large participant population to generate a formal statistics. However, we still obtained some insights from the feedback, which can help us to improve our DSVLs and tool.

The questionnaire covered different aspects of the usability, e.g. the "learnability" of the tool and user satisfaction. We prepared two sets of the questions, which were separated into two sections.

In the first section, we used ratio-based closed-end questions to obtain users' feedback for different aspects of the usability that we were interested in. For each of these questions, there are five ratio answers: "Strongly disagree", "Disagree", "Neutral", "Agree", and "Strongly Agree". We prepared six closed-ended questions which required the participants to give their opinions. Table 5 shows the closed-ended questions.

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
1.1 If features and functions provided are useful and appropriate for doing the tasks.	❒	❒	❒	❒	❒
1.2 It is easy to fix if I made a mistake.	❒	❒	❒	❒	❒
1.3 The tool provides useful and timely feedback.	❒	❒	❒	❒	❒
1.4 I find the user interface to be intuitive and easily understood.	❒	❒	❒	❒	❒
1.5 The metaphors are easy to understand and easy to read.	❒	❒	❒	❒	❒
1.6 Overall, the tool is easy to use.	❒	❒	❒	❒	❒

Table 5. Closed Ended Questions for Our Information User Test.

The first question enquires the functionality of our tool. We asked the participants "If features and functions provided are useful and appropriate for doing the tasks". The purpose of this question is to get participants' insights and the rational measurements on the functionality of the tool, and see if these functions are appropriate and sufficient for performing modelling tasks.

The second question aims to find out the users' acceptance level for mistake-recovery of our tool. We asked the participants "it is easy to fix if I made a mistake".

For the third question, we were trying to get user's responses on the feedback system of our tool, so we asked the participants if "the tool provides useful and timely feedback".

For the fourth question, we would like to measure users' experience on the user interface of the tool, so we asked the participants "I find the user interface to be intuitive and easily understood".

For the fifth question, we would like to get users' feedback on the maturity of the visual notations, so we asked participants if they find "the metaphors are easy to understand and easy to read".

For the last question, we wanted to know users' opinions on the overall usability of our tool, so we asked participants that "overall, the tool is easy to use".

Furthermore, the second section of this questionnaire was used to get participants' comments by using open-ended questions, for example, "Is there anything needs to be improved". For this section, the questions were prepared to encourage the participants to express anything they concerned or they favoured for our DSVLs and support tool. They can put any comments. Table 6 shows our open-ended questions.

1.	Is there anything that needs to be improved?
2.	What are the things you enjoyed when using the tool?
3.	Do you have any other comments?

Table 6. The Open-ended Questions of our Questionnaire.

The purpose of these open-ended questions was to explore the possibilities to enhance the DSVLs and the tool from a user's point of view. Compared with closed-ended questions, the open-ended questions are more effective to allow the participants to judge and measure the usability of our tool and the answers of these questions are more unbiased than the closed-end ones (Griffith, Cook, Guyatt, & Charles, 1999).

8.3.4 Discussion

For the informal user test (in Auckland, New Zealand), the participants' behaviours were observed. Most of the participants successfully completed the short modelling tasks. We found that using conventions in both VL design and functions design is very useful, which helped most of the participants quickly to understand the DSVLs and the modelling tool.

However, we observed that one participant struggled and failed to perform some tasks due to ignoring the conversions. Overall, they all found the visual notations are easy to understand. Most of them found the design paradigms are easy to use and understand, especially the different colour themes are most useful for them to remember the visual notations.

For the closed-ended questions, the overall feedback from the survey was very positive. All close-ended questions received rather high scores. For the open-ended questions, the participants addressed a number of aspects of its usability. Most of them suggested the visual notations are easy to understand, and the system provided feedback was very useful when they were performing the modelling tasks. The participant also suggested a few areas which need to pay more attention, for example, confusing system messages, and improvement of some of the GUI windows.

For the evaluation for the SME users, a walk-through of the DSVLs and the support tool has been demonstrated to these participants. One of the SME users, who also had the right knowledge and capabilities, was asked to perform some modelling tasks. Other participants were asked to observe and to provide comments. After that, the participants gave very positive feedback for the support of DSVLs and the modelling tool.

During the evaluation walk-through, they also identified further requirements for the workflow modelling and manipulation. They agreed that the facilitates of our modelling tool are very good. The SME participants suggested it is still doubtful whether it is possible for direct use by general SME users since it is inherently complex and comprehensive for knowledge management. They did point out that our tool allows them to achieve strategic objectives in terms of managing supply chain domain knowledge, and rather easy to understand. They all agreed the DSVLs and support tool have fulfilled the primary development goal which is the support for supply chain modelling. They also suggested the potential knowledge management functionality for future development.

For the time being, we did not carry out a large scale of usability tests. In our future work, we need to conduct more formal usability tests and evaluation which gives extensive views and feedback to improve our DSVLs and support tool.

8.4. Summary

In this chapter, we discussed a cognitive evaluation we conducted for our DSVLs and support tool, focusing on their usability. By using Green's CDs framework, we discussed the strengths and the weaknesses of our DSVLs and support tool against each of cognitive dimensions. We discussed a preliminary usability study we carried out for our DSVLs and support tool. We also discussed an industrial user evaluation that was conducted by a third party at end of the project, and discussed the findings from the users' feedback. The areas require further improvements will be addressed in our future work.

Chapter 9. Conclusion and Future Work

9.1. Introduction

In this chapter we summarise our work. Firstly, we summarise the key contributions of our work to the related research areas. Secondly, we discuss future work related to our research and tool development. Lastly, we present an overall summary for our work and research.

9.2. Contributions

Our work delivers a novel solution to support visual supply chain modeling and knowledge sharing/management. Our work makes a number of contributions across a variety research fields, including: visual language design, visualization for supply chain knowledge and knowledge management, support for supply chain modeling, implementing MDE for supply chain modeling IDE, distributed Agile (XP and SCRUM) methodologies, and Software Engineering 2.0 practices. Keys amongst these are:

- A literature review shows the state of the art of related research fields. To conduct our research in a sound theoretical framework, we carried out a literature review in terms of visual language design, supply chain modeling, knowledge management, distributed agile mythologies (focusing on XP and distributed SCRUM). The results of this literature review described the state of the art of the related research fields, e.g. challenges and solutions. The findings of this literature review not only benefit our research, but also offer valuable knowledge for other research.
- The SUDDEN visual language is a suite of visual notations, which is one of our major contributions. The goal of this visual language development was to visualize complex domain-specific concepts and knowledge against supply chain modeling, and share information among users effectively. The notations of the MaramaSUDDEN visual language have accomplished this task. In addition, our visual notation design led to the research on scientific visual notation design (Moody, 2009a) by using both the Physics of Notations framework and the CDs framework.

- The MaramaSUDDEN modeling environment – as a supply chain modeling tool, this supports the MaramaSUDDEN domain specific visual languages. The tool provides a novel perspective for computer aided visual supply chain modeling and simulation support. For example, the visual models facilitate SMEs users to plan their business activities within supply chain networks. And platform independent and machine readable knowledge repositories can be visually modified and manipulated.
- Supply chain modeling support - Cope et al (2007) identify and summarise the features of the supply chain environment, which are uncertainty, dynamic change and distributed decision making. Within the scope of our research, our DSVLs and support tool facilitate SME users against all these issues. Firstly, it provides a multi-diagram and multi-view mechanism that enables supply chain practitioners to simulate different possible solution options, and synchronize their work against existing knowledge in knowledge repositories. Users' ideas (knowledge) are visualized by the graphic models, and checked against the meta-models and knowledge repositories. In addition, its multiple possible solutions enabled functionality (e.g. multiple possible product potential decomposition) reduced the cause of the uncertainty. Therefore, the computer aided simulation models can help end users to decrease uncertainty. Secondly, a supply chain is a dynamic business environment. The computer aided simulation model is a cost-effective approach to reflect the changes of supply chains in the real world. SMEs can visually plan and coordinate their business activities and processes in computer based environments. Any changes of real world supply chains can be instantly reproduced and simulated in computer aided graphical models. Finally, supply chain partners are distributed geographically. To allow supply chain practitioners to share knowledge and interoperate together, our support tool employs MDA that ensures platform independent knowledge repositories. SMEs can use our tool visually customize, manage and manipulate knowledge repositories in Web semantic format (OWL ontology). Different applications can access and manage knowledge repositories either locally or remotely, which improves coordination and collaboration in supply chain modeling. Since OWL ontologies are portable across platforms, our tool does not need to handle any Web service issues.
- Distributed Agile practices were exercised in our GSD project. In this thesis, we presented our findings from using agile methodologies in our global software development (GSD) environment. We discussed the challenges that we were facing in GSD, for example, communication overhead, the time zone differences and project

management. In addition, we presented the limitations of some classic agile practices in GSD environment. We reported our findings from using adapted agile practices in our GSD project, e.g. Distributed SCRUM. However, we also pointed out advantages and limitations of using Distributed Agile in GSD environment, e.g. project management.

- Web 2.0 tools used to support our development process – we demonstrated how we used Web 2.0 applications to improve our communication and project management over distributed teams in the GSD environment, e.g. using a professional social network for virtual collaboration working environment. Also, we discussed how the impact of Web 2.0 technologies and applications for software engineering, and the concerns and limitations of Web2.0 in GSD context, e.g. stable Internet access required, security issues.
- Visualization of Knowledge Management – our tool is also a visual knowledge development and management IDE. Different users' knowledge is stored as ontology based repositories. The graphic and GUI based IDE facilitates users to directly manipulate, create, and edit knowledge, e.g. business processes, partner profiles, locations.

9.3. Future work

1. Data consistency and validation mechanism - since our tool provides multi-view and multi-diagram facilities and real-time/non-real-time editing supports, our tool faced a variety of challenges related to consistency issues, especially for data consistency between the modelling interface and the data repositories at non-real-time editing mode. To improve the usability and robustness of our tool, it is necessary to improve the data consistency and validation for modelling scenarios at non-real-time modelling. For example, using cache or temporary database to provide limited validation capacities when the modelling interface is not synchronised with any database.
2. Visual notation design – we used two visual notation design frameworks to implement our visual notation design focusing on the Physics of Notations framework. Due to the time constraints, some of the visual notation designs are still proof of concept only. There is much room to improve, for example, using texture to facilitate and improve the visual syntax of visual notation design. By experience and lessons gained by current stage development, we should be capable fully applying Type IV and Type V

theories (Moody, 2009a) of Physics of Notations to drive our visual design paradigms to advance for rapid SE based visual language prototyping. In the future, we need to conduct a broader evaluation to advance our visual notation design.

3. More functions to improve usability - we have developed a rich set of functions to assist our MaramaSUDDEN users. However, to manage supply chain knowledge, especially to edit ontology-based knowledge repositories, more features can be added for editing OWL repositories, for example, improvements for syntax and semantic checks, and devolved ontology functionality.

9.4. Summary

Our research has explored a number of research fields, e.g. visual language design and supply chain modelling support. We focussed on the visualisation of domain specific knowledge management for supply chain coordination and collaboration. We conducted a broad literature review and presented the challenges, problems, and the state of the art in these related research fields. Based on our literature review and interview with the stakeholders, we identified and summarised the major requirements for developing a suit of DSVLs and support tool.

After that, we discussed our design decisions and design approaches (e.g. providing visualisation support for domain-specific knowledge and knowledge management). We used both the CDs and Physics of Notations frameworks for our visual notation design, which are the building blocks of our DSVLs and support tool. We used a software engineering based MDE approach to design our modelling environment. The architecture of our tool includes four layers: an interface layer, a business logical layer, a utility layer, and a data layer. The multi-layer design ensures the data repositories are independent from any applications. We also presented a number of mechanisms we used to facilitate our users at run-time, for example, a real-time/non-real-time editing mechanism, and a consistency validation mechanism.

We then discussed our implementation decisions and choices, and the experiences and lessons learnt from using different technologies and techniques during implementation. Then, we presented the findings of using software engineering 2.0 during our development process.

Finally, we illustrated how we evaluated our DSVLs and tool by using the CDs framework and the informal user test and evaluation.

Reference

Alavi, M., & Leidner, D. (2001). Review: Knowledge Management and Knowledge Management Systems: Conceptual Foundations and Research Issues. *MIS Quarterly, 25*(1), 107-136.

Andres, H.-D. (2007). *Word of Mouth and Taste Matching: A Theory of the Long Tail*: NET Institute.

Arlow, J., & Neustadt, I. (2005). *UML 2.0 and the Unified Process: Practical Object-Oriented Analysis and Design (2nd Edition)*: Addison-Wesley Professional.

Atkinson, C. (1997). *Meta-modelling for distributed object environments.* Paper presented at the Enterprise Distributed Object Computing Workshop [1997]. EDOC '97. Proceedings. First International.

Bailey, P., Manktelow, K., & Olomolaiye, P. (2003). *Examination of the colour selection process within digital design for the Built Environment*. Paper presented at the Proceedings of the Theory and Practice of Computer Graphics 2003.

Banker, R. D., Datar, S. M., & Kemerer, C. F. (1991). A Model to Evaluate Variables Impacting the Productivity of Software Maintenance Projects. *Management Science, 37*(1), 1-18.

Barone, R., & Cheng, P. C.-H. (2004). *Representations for Problem Solving: On the Benefits of Integrated Structure*. Paper presented at the Proceedings of the Information Visualisation, Eighth International Conference.

Bezivin, J., Barbero, M., & Jouault, F. (2007). *On the Applicability Scope of Model Driven Engineering.* Paper presented at the Model-Based Methodologies for Pervasive and Embedded Software, 2007. MOMPES '07. Fourth International Workshop on.

Bickford, P. (1997). *Interface design : the art of developing easy-to-use software*. Boston, Mass.: AP Professional.

Blackwell, A. (2008). Cognitive Dimensions of Notations: Understanding the Ergonomics of Diagram Use *Diagrammatic Representation and Inference* (pp. 5-8).

Blackwell, A. F. (2006). Ten years of cognitive dimensions in visual languages and computing: Guest Editor's introduction to special issue. *Journal of Visual Languages & Computing, 17*(4), 285-287.

Bottoni, P., & Grau, A. (2004). *A Suite of Metamodels as a Basis for a Classification of Visual Languages.* Paper presented at the Visual Languages and Human Centric Computing, 2004 IEEE Symposium on.

Brown, A. (2004). An introduction to Model Driven Architecture Retrieved Oct 25, 2009, from http://www.ibm.com/developerworks/rational/library/3100.html

Brown, P. C. (1997). *Satisfying the graphical requirements of visual languages in the DV-Centro Framework*. Paper presented at the Proceedings of the 1997 IEEE Symposium on Visual Languages (VL '97).

Bull, R. I., Storey, M. A., Favre, J. M., & Litoiu, M. (2006). *An Architecture to Support Model Driven Software Visualization.* Paper presented at the Program Comprehension, 2006. ICPC 2006. 14th IEEE International Conference on.

Burke, A. E., Stel, A. J. v., Thurik, A. R., & Revision, E.-O. (2009). *Blue Ocean versus Competitive Strategy: Theory and Evidence*: Erasmus Research Institute of Management (ERIM), ERIM is the joint research institute of the Rotterdam School of Management, Erasmus University and the Erasmus School of Economics (ESE) at Erasmus Uni.

Campos, L. B., & Zorzo, S. D. (2007). *A Domain Analysis Approach for Engineering RFID Systems in Supply Chain Management.* Paper presented at the System of Systems Engineering, 2007. SoSE '07. IEEE International Conference on.

Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., & Wilkinson, K. (2004). *Jena: implementing the semantic web recommendations*. Paper presented at the Proceedings of the 13th international World Wide Web conference on Alternate track papers \& posters.

Chang, S. K. (2005). *Handbook of Software Engineering And Knowledge Engineering: Recent Advances*: World Scientific Publishing Co., Inc.

Changrui, R., Jin, D., Hongwei, D., & Wei, W. (2006). *A SCOR-Based Framework for Supply Chain Performance Management.* Paper presented at the Service Operations and Logistics, and Informatics, 2006. SOLI '06. IEEE International Conference on.

Chao, L., An, X., & Ye, L. (2009). *Knowledge Life Cycle on Semantic Web.* Paper presented at the Information Technology and Applications, 2009. IFITA '09. International Forum on.

Chatfield, D. C., Harrison, T. P., & Hayya, J. C. (2009). SCML: An information framework to support supply chain modeling. *European Journal of Operational Research, 196*(2), 651-660.

Cohen, B., & Thias, M. (2009). *The Failure of the Off-shore Experiment: A Case for Collocated Agile Teams.* Paper presented at the Agile Conference, 2009. AGILE '09.

Cohen, D., Lindvall, M., & Costa, P. (2004). An introduction to agile methods.

Cook, S., Jones, G., Kent, S., & Wills, A. (2007). *Domain-specific development with visual studio dsl tools*: Addison-Wesley Professional.

Cope, D., Fayez, M. S., Mollaghasemi, M., & Kaylani, A. (2007). *Supply chain simulation modeling made easy: an innovative approach*. Paper presented at the Proceedings of the 39th conference on Winter simulation: 40 years! The best is yet to come.

Council, S. C. (2008). SCOR Frameworks Retrieved Oct 25, 2009, from http://www.supply-chain.org/resources/scor

Cuesta, C. E., del Pilar Romay, M., de la Fuente, P., & Barrio-Solórzano, M. (2005). Architectural Aspects of Architectural Aspects (pp. 247-262).

Da-Qian, Z., & Kang, Z. (1998). *VisPro: a visual language generation toolset.* Paper presented at the Visual Languages, 1998. Proceedings. 1998 IEEE Symposium on.

Dangelmaier, W., Heidenreich, J., & Pape, U. (2005). *Supply chain management: a multi-agent system for collaborative production planning.* Paper presented at the e-Technology, e-Commerce and e-Service, 2005. EEE '05. Proceedings. The 2005 IEEE International Conference on.

Davies, J., Lytras, M., & Sheth, A. P. (2007). Guest Editors' Introduction: Semantic-Web-Based Knowledge Management. *Internet Computing, IEEE, 11*(5), 14-16.

de Castro, V., Mesa, J., Herrmann, E., & Marcos, E. (2008). *A Model Driven Approach for the Alignment of Business and Information Systems Models.* Paper presented at the Computer Science, 2008. ENC '08. Mexican International Conference on.

Deursen, A. v., Klint, P., & Visser, J. (2000). Domain-specific languages: an annotated bibliography. *SIGPLAN Not., 35*(6), 26-36.

Diehl, S. (2005). *Software visualization*. Paper presented at the Proceedings of the 27th international conference on Software engineering.

Dillon, T. S., Chang, E., & Wongthongtham, P. (2008). *Ontology-Based Software Engineering- Software Engineering 2.0.* Paper presented at the Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on.

DiMicco, J., Millen, D. R., Geyer, W., Dugan, C., Brownholtz, B., & Muller, M. (2008). *Motivations for social networking at work*. Paper presented at the Proceedings of the 2008 ACM conference on Computer supported cooperative work.

Eclipse development using the graphical editing framework and the eclipse modeling framework (2004). IBM Corp.

Evans, A. (2001). *Uml 2000 - the Unified Modeling Language: Advancing the Standard: Third International Conference, York, U. K., October 2-6, 2000, Proceedings*: Springer-Verlag New York, Inc.

Feigenbaum, B. (2006). SWT, Swing or AWT: Which is right for you? Retrieved Dec 16, 2009, from http://www.ibm.com/developerworks/grid/library/os-swingswt/

Ferguson, R., Hunter, A., & Hardy, C. (2000). MetaBuilder: The Diagrammer's Diagrammer *Theory and Application of Diagrams* (pp. 407-421).

Gabriel Alves, J., Maciel, P., & Lima, R. (2007). *Modeling and evaluation of supply chains with GSPN components*. Paper presented at the Proceedings of the 2nd international conference on Performance evaluation methodologies and tools.

Gasevic, D., Djuric, D., & Devedzic, V. (2009). *Model Driven Engineering and Ontology Development*: Springer Publishing Company, Incorporated.

Gough, C. A. D. H., Green, R., & Billinghurst, M. (2006). *Accounting for user familiarity in user interfaces*. Paper presented at the Proceedings of the 7th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: design centered HCI.

Green, T. R. G. (1989). *Cognitive dimensions of notations*. Paper presented at the Proceedings of the fifth conference of the British Computer Society, Human-Computer Interaction Specialist Group on People and computers V.

Green, T. R. G., & Petre, M. (1996). Usability Analysis of Visual Programming Environments: A Cognitive Dimensions Framework. *Journal of Visual Languages & Computing, 7*(2), 131-174.

Greenfield, J. (2004). The Case for Software Factories Retrieved Oct 25, 2009

Greenfield, J., & Short, K. (2003). *Software factories: assembling applications with patterns, models, frameworks and tools*. Paper presented at the Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications.

Griffith, L. E., Cook, D. J., Guyatt, G. H., & Charles, C. A. (1999). Comparison of Open and Closed Questionnaire Formats in Obtaining Demographic Information From Canadian General Internists. *Journal of Clinical Epidemiology, 52*(10), 997-1005.

Grundy, J., Hosking, J., Huh, J., & Li, K. N.-L. (2008). *Marama: an eclipse meta-toolset for generating multi-view environments*. Paper presented at the Proceedings of the 30th international conference on Software engineering.

Grundy, J., Hosking, J., Huh, J., & Na-Liu Li, K. (2008). *Marama*. Paper presented at the Software Engineering, 2008. ICSE '08. ACM/IEEE 30th International Conference on.

Grundy, J., Hosking, J., Nianping, Z., & Na, L. (2006, 18-22 Sept. 2006). *Generating Domain-Specific Visual Language Editors from High-level Tool Specifications*. Paper presented at the Automated Software Engineering, 2006. ASE '06. 21st IEEE/ACM International Conference on.

Hai-Ying, C., Stucky, W., & Yan-Bin, J. (2008). *Using XML nets and grid services to support SCOR*. Paper presented at the Machine Learning and Cybernetics, 2008 International Conference on.

Hole, S., & Moe, N. B. (2008). A Case Study of Coordination in Distributed Agile Software Development *Software Process Improvement* (pp. 189-200).

Horn, R. (1999). *Visual Language: Global Communication for the 21st Century*: {Macrovu Inc.}.

Hosking, J. (2009). *Supporting model driven engineering using the Marama meta toolset.* Paper presented at the Enterprise Distributed Object Computing Conference Workshops, 2009. EDOCW 2009. 13th.

Hosking, J., Mehandjiev, N., & Grundy, J. (2008). *A domain specific visual language for design and coordination of supply networks*. Paper presented at the Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing.

Hossain, E., Babar, M. A., & Hye-young, P. (2009). *Using Scrum in Global Software Development: A Systematic Literature Review.* Paper presented at the Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on.

Hossain, E., Babar, M. A., Hye-young, P., & Verner, J. (2009, 1-3 Dec. 2009). *Risk Identification and Mitigation Processes for Using Scrum in Global Software Development: A Conceptual Framework.* Paper presented at the Software Engineering Conference, 2009. APSEC '09. Asia-Pacific.

International Organization for Standardization (n.d.). ISO 9241-11 (Vol. 2009).

Jian, M., Shijin, G., & Liyi, Z. (2008). *The Community Website Design Based on Complementary Advantages of Web2.0 and Web1.0.* Paper presented at the Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on.

Jiang, T., & Tianfield, H. (2006). *Literature Review Upon Multi-Agent Supply Chain Management.* Paper presented at the Machine Learning and Cybernetics, 2006 International Conference on.

Jin, D., Hongwei, D., Changrui, R., & Wei, W. (2006). *IBM SmartSCOR - A SCOR Based Supply Chain Transformation Platform Through Simulation and Optimization Techniques.* Paper presented at the Simulation Conference, 2006. WSC 06. Proceedings of the Winter.

Johannsen, G. (2009). Design of Visual and Auditory Human-Machine Interfaces with User Participation and Knowledge Support *Industrial Engineering and Ergonomics* (pp. 499-509).

Julsrud, T. E., & Schiefloe, P. M. (2007). The development, distribution and maintenance of trust in distributed work groups: a social network approach. *Int. J. Netw. Virtual Organ., 4*(4), 351-368.

Kasi, V. (2005). *Systemic Assessment of SCOR for Modeling Supply Chains.* Paper presented at the System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on.

Kontio, J., Hoglund, M., Ryden, J., & Abrahamsson, P. (2004). *Managing Commitments and Risks: Challenges in Distributed Agile Development*. Paper presented at the Proceedings of the 26th International Conference on Software Engineering.

Korkala, M., & Abrahamsson, P. (2007). *Communication in Distributed Agile Development: A Case Study.* Paper presented at the Software Engineering and Advanced Applications, 2007. 33rd EUROMICRO Conference on.

Lee, S., & Yong, H.-S. (2009). Distributed agile: project management in a global environment. *Empirical Software Engineering.*

Li, J., Psarrou, A., Sanxing, C., & Yajing, C. (2009). *Trends on Interactive Platforms for Social Media through Web2.0.* Paper presented at the Management and Service Science, 2009. MASS '09. International Conference on.

Li, L., Hosking, J., & Grundy, J. (2008). *MaramaEML: An Integrated Multi-View Business Process Modelling Environment with Tree-Overlays, Zoomable Interfaces and Code*

Generation. Paper presented at the Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering.

Li, P., & Wohlstadter, E. (2008). *View-based maintenance of graphical user interfaces*. Paper presented at the Proceedings of the 7th international conference on Aspect-oriented software development.

Liew, P., Kontogiannis, K., & Tong, T. (2004). *A framework for business model driven development.* Paper presented at the Software Technology and Engineering Practice, 2004. STEP 2004. The 12th International Workshop on.

Lippert, M., Wolf, H., & Roock, S. (2002). *Extreme Programming in Action: Practical Experiences from Real World Projects*: John Wiley \& Sons, Inc.

Malone, T. W., & Crowston, K. (2003). *Toward an interdisciplinary theory of coordination*: Massachusetts Institute of Technology (MIT), Sloan School of Management.

Malone, T. W., Crowston, K., & Herman, G. A. (2003). *Organizing Business Knowledge: The MIT Process Handbook*: MIT Press.

Maplesden, D., Hosking, J., & Grundy, J. (2001). *A Visual Language for Design Pattern Modelling and Instantiation*. Paper presented at the Proceedings of the IEEE 2001 Symposia on Human Centric Computing Languages and Environments (HCC'01).

Martin, D., Burstein, M., Mcdermott, D., Mcilraith, S., Paolucci, M., Sycara, K., et al. (2007). Bringing Semantics to Web Services with OWL-S. *World Wide Web, 10*(3), 243-277.

Maurer, F., & Melnik, G. (2006). *Agile methods: moving towards the mainstream of the software industry*. Paper presented at the Proceedings of the 28th international conference on Software engineering.

Maurizio, B., & Mariagrazia, M. (2008). Trends In Modelling Supply Chain And Logistic Networks.

McDonald, D. W. (2003). *Recommending collaboration with social networks: a comparative evaluation*. Paper presented at the Proceedings of the SIGCHI conference on Human factors in computing systems.

Mehandjiev, N. D., Stalker, I. D., & Carpenter, M. R. (2009). Recursive Construction and Evolution of Collaborative Business Processes *Business Process Management Workshops* (pp. 573-584).

Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Comput. Surv., 37*(4), 316-344.

Min, H., & Zhou, G. (2002). Supply chain modeling: past, present and future. *Comput. Ind. Eng., 43*(1-2), 231-249.

Moe, N. B., & Aurum, A. (2008). *Understanding Decision-Making in Agile Software Development: A Case-study*. Paper presented at the Software Engineering and Advanced Applications, 2008. SEAA '08. 34th Euromicro Conference.

Moody, D. (2009a). The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *Software Engineering, IEEE Transactions on, 35*(6), 756-779.

Moody, D. (2009b). *Theory development in visual language research: Beyond the cognitive dimensions of notations.* Paper presented at the Visual Languages and Human-Centric Computing, 2009. VL/HCC 2009. IEEE Symposium on.

Moretti, G. S., & Lyons, P. J. (2005). *Controlling the complexity of grouped items in colour interfaces*. Paper presented at the Proceedings of the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: making CHI natural.

Mowshowitz, A. (1997). Virtual organization. *Commun. ACM, 40*(9), 30-37.

Myers, C., & Baniassad, E. (2009). *Silhouette: visual language for meaningful shape*. Paper presented at the Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications.

N. Hari Narayanan, & Roland, H. (1998). Visual language theory: towards a human computer interaction perspective *Visual language theory* (pp. 87-128): Springer-Verlag New York, Inc.

Newkirk, J. (2002). *Introduction to agile processes and extreme programming.* Paper presented at the Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on.

Nielsen, J. (1993). *Usability engineering*. Boston: Academic Press.

Northover, S., & Wilson, M. (2004). *Swt: the standard widget toolkit, volume 1*: Addison-Wesley Professional.

OMG (2009). OMG Model Driven Architecture Retrieved Oct 25, 2009, from http://www.omg.org/mda/

Paasivaara, M., Durasiewicz, S., & Lassenius, C. (2008). *Distributed Agile Development: Using Scrum in a Large Project.* Paper presented at the Global Software Engineering, 2008. ICGSE 2008. IEEE International Conference on.

Paasivaara, M., Durasiewicz, S., & Lassenius, C. (2009). *Using Scrum in Distributed Agile Development: A Multiple Case Study.* Paper presented at the Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on.

Paetsch, F., Eberlein, A., & Maurer, F. (2003). *Requirements engineering and agile software development.* Paper presented at the Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on.

Pastore, S. (2008). *Social networks, collaboration and groupware software for the scientific research process in the web 2.0 world*. Paper presented at the Proceedings of the 7th WSEAS International Conference on Artificial intelligence, knowledge engineering and data bases.

Reifer, D. J. (2002). How Good are Agile Methods? *IEEE Softw., 19*(4), 16-18.

Reinhard, T., Seybold, C., Meier, S., Glinz, M., & Merlo-Schett, N. (2006). *Human-Friendly Line Routing for Hierarchical Diagrams.* Paper presented at the Automated Software Engineering, 2006. ASE '06. 21st IEEE/ACM International Conference on.

Ribino, P., Oliveri, A., Re, G. L., & Gaglio, S. (2009). *A Knowledge Management System Based on Ontologies.* Paper presented at the New Trends in Information and Service Science, 2009. NISS '09. International Conference on.

Richards, D. (2009). A social software/Web 2.0 approach to collaborative knowledge engineering. *Inf. Sci., 179*(15), 2515-2523.

Rundensteiner, E. A. (1992). *MultiView:A Methodology for Supporting Multiple Views in Object-Oriented Databases* Paper presented at the Proceedings of the 18th International Conference on Very Large Data Bases (VLDB'92, Vancouver, British Columbia, Canada.

Sanderson, S. (2008). *ASP.NET MVC Framework Preview*: Apress.

Sarker, S., Lau, F., & Sahay, S. (2000, 4-7 Jan. 2000). *Building an inductive theory of collaboration in virtual teams: an adapted grounded theory approach.* Paper presented at the System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on.

Schmidt, D. C. (2006). Guest Editor's Introduction: Model-Driven Engineering. *Computer, 39*(2), 25-31.

Schneider, F., Feldmann, A., Krishnamurthy, B., & Willinger, W. (2009). *Understanding online social network usage from a network perspective*. Paper presented at the

Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference.

Schwaber, K. (2004). *Agile Project Management With Scrum*: Microsoft Press.

Selker, T., & Koved, L. (1988). *Elements of visual language.* Paper presented at the Visual Languages, 1988., IEEE Workshop on.

Seo, H.-S., Chung, I. S., Kim, B. M., & Kwon, Y. R. (2001). *The Design and Implementation of Automata-based Testing Environment for Java Multi-thread Programs*. Paper presented at the Proceedings of the Eighth Asia-Pacific on Software Engineering Conference.

Sprinkle, J., & Karsai, G. G. (2004). A domain-specific visual language for domain model evolution. *Journal of Visual Languages & Computing, 15*(3-4), 291-307.

Srinivasan, J., & Lundqvist, K. (2009). *Using Agile Methods in Software Product Development: A Case Study*. Paper presented at the Information Technology: New Generations, 2009. ITNG '09. Sixth International Conference on.

Steffen, B. (2009). *Keynote: Continuous Model Driven Engineering.* Paper presented at the Engineering of Complex Computer Systems, 2009 14th IEEE International Conference on.

Sun Microsystems, I. (2002). Abstract Window Toolkit (AWT) Retrieved Nov 20, 2009, from http://java.sun.com/j2se/1.4.2/docs/guide/awt/

Sun Microsystems, I. (n.d.). Processes and Threads Retrieved Dec 18, 2009, from http://java.sun.com/docs/books/tutorial/essential/concurrency/procthread.html

Sureshchandra, K., & Shrinivasavadhani, J. (2008). *Adopting Agile in Distributed Development.* Paper presented at the Global Software Engineering, 2008. ICGSE 2008. IEEE International Conference on.

Tavassoli, S., Sardashti, M., & Toussi, N. (2009). *Supply chain management and information technology support.* Paper presented at the Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on.

Therrien, E. (2008). *Overcoming the Challenges of Building a Distributed Agile Organization.* Paper presented at the Agile, 2008. AGILE '08. Conference.

Tolvanen, J.-P., & Kelly, S. (2009). *MetaEdit+: defining and using integrated domain-specific modeling languages*. Paper presented at the Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications.

US Homeland Security (2008). Homeland Security Advisory System--Guidance for Federal Departments and Agencies Retrieved 11 Aug, 2009, from http://www.dhs.gov/files/programs/gc_1156876241477.shtm

V ćtor Bl ázquez, I. S., Martin Carpenter, G., & Weichhart (2008). *Detailed System Architecture* (The SUDDEN project No. Deliverable 4.2).

van Lamsweerde, A. (2001). *Goal-oriented requirements engineering: a guided tour*. Paper presented at the Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on.

Varma, V. (2007). Use of Ontologies for Organizational Knowledge Management and Knowledge Management Systems *Ontologies* (pp. 21-47).

Vax, M., & Michaud, S. (2008). *Distributed Agile: Growing a Practice Together*. Paper presented at the Agile, 2008. AGILE '08. Conference.

Venkatesh, R., Bhaduri, P., & Joseph, M. (2001). *Formalizing models and meta-models for system development extended abstract.* Paper presented at the Software Engineering Conference, 2001. APSEC 2001. Eighth Asia-Pacific.

. *Visual language theory*. (1998): Springer-Verlag New York, Inc.

Wilson, L. T., & Snyder, C. A. (1999). Knowledge management and IT: how are they related. *IT Professional, 1*(2), 73-75.

Wu, C. T. (2001). *An Introduction to Object-Oriented Programming with Java*: McGraw-Hill Pub. Co.

Wu, J., Ulieru, M., Cobzaru, M., & Norrie, D. (2000). *Supply chain management systems: state of the art and vision.* Paper presented at the Management of Innovation and Technology, 2000. ICMIT 2000. Proceedings of the 2000 IEEE International Conference on.

Yang, S., DeKoven, E., & Zloof, M. (1996). *Design benchmarks for VPL static representations.* Paper presented at the Visual Languages, 1996. Proceedings., IEEE Symposium on.

Ye, Y., Yang, D., Jiang, Z., & Tong, L. (2008). Ontology-based semantic models for supply chain management. *The International Journal of Advanced Manufacturing Technology, 37*(11), 1250-1260.

Yue-Shan, C., Lo, W., Chii-Jet, W., Shyan-Ming, Y., & Liang, D. (1998). *Design and implementation of multi-threaded object request broker.* Paper presented at the Parallel and Distributed Systems, 1998. Proceedings., 1998 International Conference on.

Zaihisma Che, C., & Abdullah, R. (2008). *Ontology-based Semantic Web services framework for knowledge management system.* Paper presented at the Information Technology, 2008. ITSim 2008. International Symposium on.

Zhang, K., Zhang, D. Q., & Cao, J. (2001). Design, construction, and application of a generic visual language generation environment. *Software Engineering, IEEE Transactions on, 27*(4), 289-307.

Zhifeng, Y. (2001). *Hidden Dependencies in Program Comprehension and Change Propagation.*

Appendix A – Use Cases

1) Use Case: Select & Set a database Specification

Use Case: Select & Set database
ID: 3.4.1.3.1
Description: To select a valid data repository, and set up a connection with the visual modelling interface.
Primary actors: User
Secondary actors: None
Preconditions: None
Main flow 1. The user selects “Select & Set database”. 2. The system asks the user to select a valid database. 3. The user browses and selects a valid database. 4. The system checks if it is a valid database. 4.1 The system sets the selected database. 4.2 The system tells the user the selected file is set successfully for further use. 5. Else The system goes back to step 2.
Post conditions: A valid data repository has been selected and set successfully.
Alternative flows:

None.

2) Use Case: Select & Set Translation file Specification

Use Case: Select & Set a Translation File
ID: 3.4.1.3.2
Description: The User selects and sets a valid translation file.
Primary actors: User
Secondary actors: None
Preconditions: None
Main flow 1. The user selects “Select a Translation File”. 2. The system asks the user to select a valid Translation file. 3. The user browses and selects a file. 4. The system checks if it is a valid Translation file. 4.1 The system sets the selected Translation file. 4.2 The system tells the user the selected file is set successfully. 5. Else The system goes back to step 2.
Post conditions: A valid translation file has been set successfully.
Alternative flows: None

3) Use Case: Search Class Specification

Use Case: Search Class
ID: 3.4.1.3.3
Description: The User searches information for a given class name in the database.
Primary actors: User
Secondary actors: None
Preconditions: A valid connection to a valid data repository.
Main flow 1. The user selects “Search Class”. 2. The system checks if a valid database is set. 3. If there is no valid database to search. 3.1 The system tells the user that the database is not set. 3.2 The user will be redirected to the “Select&Set a Database” use case. 4. The system asks the user to input a class name. 5. The system searches for classes that match the user’s input. 6. If the system finds the class which matching the criteria. 6.1 The system displays a list of instances and relationships if the class is found. 7. Else the system tells the user there is no matched class is found.
Post conditions: Return information to users if there are available records in the database.
Alternative flows: Select&Set a Database

4) Use Case: Link Instance Specification

Use Case: Link Instance
ID: 3.4.1.3.4
Description: 1. To link the visual icon to an existing instance in the data, 2. or to create a new instance for a existing class/subclass, 3. or to create a new instance and create a new class or subclass.
Primary actors: User
Secondary actors: None
Preconditions: A valid connection to a valid data repository. & The modelling. environment is In non-real-time editing mode.
Main flow 1. The user selects “Link Instance ”. 2. The system checks if a valid database is selected. 3. If there is no valid database is set. 3.1 The system tells the user that the database is not set. 3.2 The user will be redirected to Select&Set a Database” use case. 4. The system check the selected notation type with the function type. 4.1 The system retrieves and displays all “XX” type related classes, instances and relationships from the ontology file. 4.2 The user browses class types. 4.3 If the user selects a existing classtype/subclasstype. 4.3.1 The system displays the related instances and relationships

to this class.

4.3.2 If the user selects a existing instance.

4.3.2.1 if user saves the selection

4.3.2.1 The system links the selected notation to this instance.

4.3.3 If the user enter a new instance name

4.3.3.1 if the name is valid and if the user saves the selection.

4.3.3.1.1 a new instance is created under the selected class type.

4.3.3.1.2 The system links the selected notation to this new instance.

4.3.4 If the user does not select or enter a new name

4.3.4.1 if the user saves the selection

4.3.4.1.1 The system creates an instance with a default name.

4.3.4.1.2 The system links the selected notation to this new instance.

4.4 If the user selects a other type.

4.4.1 If the user enter a new class name and an instance name.

4.3.3.1 if the names are valid

4.4.1.1.1 a new instance is created under the selected class type.

4.4.1.1.2 The system links the selected notation to this new instance.

4.4.1.1.3 The system tells the user the action is success.

4.4.2 If the user enters a valid new class name only

4.4.2.1.1 The system creates an instance with a default name.

4.4.2.1.2 The system links the selected notation to this new instance. 4.4.2.1.3 The system tells the use the action is success. 4.4.3 If the user enters no valid name for class type 4.4.3.1 The system asks the user to input a valid class name.
Post conditions: 1. The visual icon is successfully linked to an instance in the data; 2 a new instance is created under the existing class/subclass, and linked with the visual icon successfully; 3. A new instance and a new class are created in the database, and the new instance is linked to the visual icon.
Alternative flows: Select&Set a Database

5) Use Case: Delete Instance Specification

Use Case: Delete Instance
ID: 3.4.1.3.5
Description: The User deletes an instance from the database.
Primary actors: User
Secondary actors: None
Preconditions: A valid connection to a valid data repository. & The modelling environment is In non-real-time editing mode.

Main flow 1. The user selects “Deletes Instance” 2. The system checks if a valid database is set. 3. If there is no valid database is set. 3.1 The system tells the user that the database is not set. 3.2 The user will be redirected to “Select&Set” database use case. 4. The system check if the instance is valid and if there is any constraints to remove the instance from the ontology. 4.1 The system removes the instance in the database and removes the visual notation. 4.2 The system removes the related tuples in the database if they are applicable. 4.3 The system tells the action is success. 5. Else, the system indicates the user action failed.
Post conditions: The system has removed the instance from the database, and removed the visual icon from the visual modelling environment.
Alternative flows: Select&Set a Database

6) Use Case: CreateTupe Specification

Use Case: CreateTuple
ID: 3.4.1.3.6
Description: The User creates a tuple between two valid instances in the database.
Primary actors: User
Secondary actors:

None
Preconditions: A valid connection to a valid data repository. & The modelling environment is In non-real-time editing mode.
Main flow 1. The user selects “CreateTuple”. 2. The system checks if a valid database is selected. 3. If there is no valid database is set. 3.1 The system tells the user that the database is not set. 3.2 The user will be redirected to “Select&Set” database use case. 4. The system check if the instances are valid and if there is any constraints to remove the instance from the ontology. 4.1 The system creates a tuple between the two valid instances. 4.2 The system tells the user that the action is success. 5. Else the system indicates the user action failed.
Post conditions: The system successfully created a tuple in the database.
Alternative flows: Select&Set a Database

7) Use Case: Remove Tuple Specification

Use Case: RemoveTuple
ID: 3.4.1.3.7
Description: The User removes a tuple from the visual modelling interface and the database.
Primary actors: User

Secondary actors: None
Preconditions: A valid connection to a valid data repository, & the modelling environment is in non-real-time editing mode.
Main flow 1. The user selects “RemoveTuple”. 2. The system checks if a valid database is set. 3. If there is no valid database is 3.1 The system tells the user that the database is not set. 3.2 The user will be redirected to “select & set” database use case. 4. The system check if the instances and the tuple are valid if there is any constraints to remove the instance from the ontology. 4.1 The system removes the tuple from the ontology file, and removes the visual notation. 4.2 The system tells the action is success. 5. Else the system indicates the user action failed.
Post conditions: The system has removed the tuple from the database and removes the visual representation of this tuple from the visual modelling environment.
Alternative flows: Select&Set a Database

8) Use Case: Open Link Specification

Use Case: Open Link
ID: 3.4.1.3.8
Description: The User opens a hyperlink in a MaramaSUDDEN browser

Primary actors: User
Secondary actors: None
Preconditions: A pre-defined hyperlink is pre-defined.
Main flow 1. The user selects “Open Link” 2. The system checks if the link is valid. 2.1 The system opens the hyperlink in a Web browser. 2.2 The system tells the user that the action is success. 3. Else the system indicates the user action failed.
Post conditions: None.
Alternative flows: None

9) Use Case: OpenAlternativeView Specification

Use Case: OpenView
ID: 3.4.1.3.9
Description: The User opens a modularized object in a separate view.
Primary actors: User
Secondary actors: None

Preconditions: A valid path to an alternative view is pre- defined.
Main flow 1. The user selects “OpenAlternativeView”. 2. The system check if there is a valid alternative diagram or decomposition view available. 2.1 The system opens the alternative diagrams. 3. Else the system indicates the user action failed.
Post conditions: A different view/diagram has been opened.
Alternative flows: None

10) Use Case: Browse Ontology Specification

Use Case: Browse Ontology
ID: 3.4.1.3.10
Description: The User browses information from the database.
Primary actors: User
Secondary actors: None
Preconditions: A valid connection to a valid data repository.
Main flow

1. The user selects “Browse Ontology ”. 2. The system checks if a valid database is set. 3. If there is no valid database is set. 3.1 The system tells the user that the database is not set. 3.2 The user will be redirected to the alternative use case. 4. The system retrieves and displays all classes and subclasses of the ontology. 5. The user browses class types and selects class type. 6. The system retrieves the instances for the selected class. 7. The user selects a instance. 8. The system retrieves the relationships for the selected instance.
Post conditions: None.
Alternative flows: None.

11) Use Case: Unify Shape Size Specification

Use Case: Unify Shape Size
ID: 3.4.1.3.11
Description: A valid connection to a valid data repository.
Primary actors: User
Secondary actors: None
Preconditions: In a valid Processes/Product/ASN type diagram.

Main flow 1. The user selects “UnifyShapeSize ”. 2. The system unified the size of the visual notations.
Post conditions: The size of the visual icons in the modelling environment has set to the default .
Alternative flows: None

12) Use Case: Switch real-time/non-real-time editing modes Specification

Use Case: Switch Real- Time/Non-Real-Time
ID: 3.4.1.3.12
Description: The User switches the editing modes of the visual modelling environment between the real-time and non-real-time modes. In real-time mode, the data communication with the database is synchronised, in non-real-time mode, it is not.
Primary actors: User
Secondary actors: None
Preconditions: In a valid Processes/Product/ASN type diagram
Main flow 1. The user selects “switch modes”. 2. The system asks the user to choose a mode. 3. The user selects one of two modes.

3.1 The system sets the system mode to the user's selection. 4 If the user does not select a mode. 4.1 The system sets the mode to default.
Post conditions: The editing mode of the visual environment has been set based on users' selection.
Alternative flows: None

13) Use Case: Save All Specification

Use Case: SaveAll
ID: 3.4.1.3.13
Description: The User saves all his/her work into the database.
Primary actors: User
Secondary actors: None
Preconditions: A connection to a valid the database .
Main flow 1. The user selects "SaveAll". 2. The system checks if a valid database is set. 3. If there is no valid database is set.

3.1 The system tells the user that the database is not set. 3.2 The user will be redirected to the alternative use case. 4. The system saves all valid instances and relationships into an ontology file. 5. The system tells the user the action is success.
Post conditions: The system has saved user's work into the database.
Alternative flows: None