

# **An Integrated Visual Approach for Business Process Modelling**

A thesis presented in partial fulfilment of the  
requirements for the degree of

**Doctor of Philosophy**

in

**Computer Science**

at the University of Auckland, New Zealand

**Lei Li**

**2010**

# ABSTRACT

---

Current approaches to modelling complex business processes fail to scale to large organizations. Key issues are “cobweb” and “labyrinth” problems exhibited by conventional box and line metaphors and large numbers of hidden dependencies introduced by compartment-based modularity. They require long term memory of users; have restrictions on expressiveness; and lack multiple levels of abstractions (Schnieders and Puhlmann 2005; Zhu and Grundy et al, 2007). Splitting up diagrams, even with good tool support, leads to implicit relationships among diagrams and navigational difficulties (Recker and Niehaves 2008; Grundy and Hosking et al, 2006).

Our earlier work (Anderson and Apperley 1990; Phillips 1995; Li and Phillips et al, 2004) on modelling complex user interfaces and their behaviour with visual dialogue notations demonstrated that a tree-based overlay structure can effectively mitigate these complexity problems. In addition, trees support rapid navigation, elision and automatic layout in ways difficult to achieve with graph-based approaches. This research is to develop an integrated visual approach for business process modelling. It includes a novel tree-overlay based visual notation (EML) and its integrated support environment (MaramaEML) to supplement and integrate with existing visual modeling solutions.

In EML, complex business architectures are represented as service trees and business processes are modelled as process overlay sequences on the service trees. By combining these two mechanisms EML gives users a clear overview of a whole enterprise system with business processes modelled by overlays on the same view.

MaramaEML is developed using the Eclipse-based Marama framework. It integrates EML and BPMN to provide high-level business service modelling. It supports automatic BPEL code generation from the graphical representations and facilitates process code validation using LTSA. It also provides a distortion-based fisheye and zooming function to enhance complex diagram navigation ability.

# ACKNOWLEDGEMENT

---

I am in the fortunate position of working and living with many wonderful and inspiring people. I am delighted to have the opportunity to thank them for their support, advice and friendship.

First, my sincere gratitude to my supervisors --- John Hosking and John Grundy. I have received stimulation, motivation, inspiration, optimism, criticism, guidance, and so many little things that have made my PhD research both a challenging and rewarding journey. I am more than grateful for being allowed to work with both truly great academics. I am very proud for having the opportunity to work with them, and received and taken advantage from it. The thesis would not be there without them to whom I look up to and whose advice and friendship means much to me.

A very big thanks to my father and mother, who have always believed in me, encouraged me, critically examined my development, and instilled in me a desire to identify, exploit, use and expand my skills and abilities. I have come to understand how important education, ethics and character are in growing up and becoming a better person.

Very special thanks to my loving Karen and little Teresa. I am very grateful for all the encouragement, love, support and motivation I have received from you both. You made me very aware of the importance of the feeling that only family can establish and which they sustain throughout all obstacles. Family is everything.

Last, but most important, to my grandfather. I know you are watching me in the heaven. Miss you a lot...

# CONTENTS

---

<b>ABSTRACT</b> -----	II
--	
<b>ACKNOWLEDGEMENT</b> -----	III
-----	
<b>CONTENTS</b> -----	IV
-----	
<b>LIST OF FIGURES</b> -----	VII
-----	
<b>LIST OF TABLES</b> -----	X
-----	
<b>CHAPTER 1 – INTRODUCTION</b> -----	1
-----	
1.1 Motivation -----	2
1.2 Research Targets -----	8
1.3 Thesis Structure -----	9
1.4 Publications -----	10
<b>CHAPTER 2 – LITERATURE REVIEW</b> -----	12
-----	
2.1 Business Process Modelling -----	13
2.2 Entity-Relationship (ER) Method and ER Diagram -----	16
2.3 Data Flow Method and Data Flow Diagrams -----	20
2.4 Workflow Method and BPMN -----	23
2.5 Integrated Modelling Method and UML -----	26
2.6 Aspect-Oriented Method and AspectM -----	33
2.7 Form -Oriented Method and Form Chart Diagram -----	37
2.8 Other Domain Specific Modeling Languages -----	40
2.9 Business Process Execution Language -----	41
2.10 Discussion and Summary -----	46
<b>CHAPTER 3 - ENTERPRISE MODELLING LANGUAGE</b> -----	51
-----	
3.1 EML Overview -----	51
3.2 Service Tree Structure -----	54
3.2.1 Service / Sub-Service -----	54
3.2.2 Operation -----	59
3.2.3 Tree Layout -----	62
3.2.4 Elision -----	66
3.2.5 Service Reuse -----	68

3.3	Process Overlay -----	70
3.3.1	Process Start -----	70
3.3.2	Process End -----	72
3.3.3	Data Encapsulation -----	75
3.3.4	Process Flow -----	75
3.3.5	Business Process Layer -----	77
3.4	Exception Handler Overlay -----	80
3.4.1	Failure Handling Notation -----	81
3.4.2	Exception Flow -----	83
3.4.3	Exception Layer -----	85
3.5	Dependency / Trigger -----	88
3.5.1	Trigger Flow -----	89
3.5.2	Trigger Overlay -----	91
3.6	Iteration -----	92
3.7	Condition -----	95
3.8	Summary -----	97

**CHAPTER 4 - EML MODELLING TOOL IMPLEMENTATION ----- 98**

4.1	Introduction -----	98
4.2	Pounamu EML Tool -----	99
4.3	Marama EML Tool -----	104
4.3.1	Structural Backbone -----	106
4.3.2	Tool Project -----	107
4.3.3	Metamodel -----	108
4.3.4	Shapes and Connectors -----	110
4.3.5	View Types -----	111
4.3.6	Model Projects -----	113
4.3.7	Behaviours -----	114
4.3.8	Service Tree Structure -----	114
4.3.9	Overlay -----	116
4.3.10	Code Generation -----	117
4.3.11	Zoomable View -----	118
4.3.12	Fisheye View -----	120
4.4	Integration -----	123
4.5	Summary -----	125

**CHAPTER 5 – CASE STUDY**

5.1	University Enrollment System Example -----	127
5.2	Service Tree Modeling -----	128
5.3	Overlay for Processes, Exceptions and Triggers -----	132

5.4	<i>BPMN Integration</i>	135
5.5	<i>BPEL Generation and LTSA Validation</i>	140
5.6	<i>Zoomable and Fisheye Views</i>	142
5.7	<i>Deployment</i>	144
5.8	<i>Summary</i>	147
<b>CHAPTER 6 – EVALUATION</b>		149
6.1	Evaluation Mechanisms Overview	149
6.2	Cognitive Dimensions for Early Validation and Design Assistance	150
6.2.1	<i>Consistency</i>	151
6.2.2	<i>Visibility &amp; Juxtaposability</i>	153
6.2.3	<i>Premature Commitment</i>	154
6.2.4	<i>Hidden Dependencies</i>	154
6.2.5	<i>Error Proneness</i>	157
6.2.6	<i>Abstraction</i>	158
6.2.7	<i>Secondary Notation</i>	160
6.2.8	<i>Closeness of mapping</i>	160
6.2.9	<i>Diffuseness</i>	161
6.2.10	<i>Hard mental operations</i>	161
6.3	Early Evaluation with Experienced Tool Developers	162
6.3.1	<i>Evaluation Environment</i>	163
6.3.2	<i>Brief Evaluation Process</i>	166
6.3.3	<i>Informal Evaluation Results</i>	166
6.3.4	<i>Improvements from the second evaluation</i>	167
6.4	Large Formal Evaluation	167
6.4.1	<i>Participant recruitment</i>	168
6.4.2	<i>Evaluation Approach</i>	170
6.4.3	<i>Brief Evaluation Schedule</i>	171
6.4.4	<i>Data Analysis</i>	173
6.4.5	<i>Improvements from the formal evaluation</i>	177
6.5	Summary	177
<b>CHAPTER 7 – CONCLUSION</b>		178
7.1	Research Output	178
7.2	Future Work	180
7.3	Conclusion	183
<b>REFERENCES</b>		185
<b>APPENDIX</b>		208

## **LIST OF FIGURES**

---

Figure 1.1 Part of a BPMN specification of the Enrolment System -----	5
Figure 1.2 Using Lean Cuisine to Present Style example -----	7
Figure 2.1 Enterprise Level Business Process Modelling Framework Overview	15
Figure 2.2 ER Diagram Example -----	17
Figure 2.3 Entity-Relationship Modeler Usage Example -----	19
Figure 2.4 Data Flow Diagram Example for a Book Order System -----	21
Figure 2.5 JUDE Data Flow Diagram Tool -----	22
Figure 2.6 BPMN Example Diagram -----	25
Figure 2.7 WebSphere Business Modeler -----	26
Figure 2.8 Activity Diagram to Model Process Order -----	30
Figure 2.9 Eriksson Penker Business Extension Structure -----	31
Figure 2.10 Sell Books Process using Eriksson Penker Approach -----	32
Figure 2.11 ArgoUML Software Tool -----	33
Figure 2.12 Major Components of Aspect Oriented Modeling -----	34
Figure 2.13 AspectM Notations and XML forms -----	36
Figure 2.14 Athene for Aspect Oriented Modeling -----	37
Figure 2.15 Form Chart Notation Example -----	39
Figure 2.16 Mapping between BPEL and WSDL -----	42
Figure 3.0 The EML meta-model -----	53
Figure 3.1 EML Service / sub-Service -----	54
Figure 3.2 EML Operation -----	60
Figure 3.3 Example EML Tree Structure -----	64
Figure 3.4 (a): Extended Customer Service; -----	67
(b): Collapsed Customer Service	
Figure 3.5 (a): Define a Reusable Service; -----	69
(b): Use Reusable sub-Service in Hotel Service	
Figure 3.6 (a): Process Start without Conditions; -----	71

(b) Process Start with Conditions	
Figure 3.7 Process End Notation -----	73
Figure 3.8 (a): A Process Flow with Sequence ID and Data; -----	
(b): A Default Process Flow	75
Figure 3.9 Travel Booking Process Overlay -----	78
Figure 3.10 (a): Failure Handling Notations in Operations; -----	
(b): Failure Handling Notation in Service	82
Figure 3.11 (a): An Exception Flow with sequence ID and Data; -----	
(b): A Default Exception Flow	84
Figure 3.12 Hotel Room Booking Exception Handler Overlay -----	86
Figure 3.13 (a): A Trigger Flow with sequence ID and Data; -----	
(b): A Default Trigger Flow	89
Figure 3.14 Make Payment Process with Triggers -----	91
Figure 3.15 (a): Loop in Process Overlay with Single Activity;	
(b): Loop in Trigger Overlay with Two Operations; -----	93
©: Loop in Exception Handler Overlay with Three Operations	
Figure 3.16 Conditions in Process Overlay -----	96
Figure 4.1 The initial exploration of EML using Pounamu -----	100
Figure 4.2 Construction of the EML metamodel in Pounamu -----	101
Figure 4.3 Construction of the EML shapes in Pounamu -----	102
Figure 4.4 Construction of the EML connectors in Pounamu -----	102
Figure 4.5 Construction of the EML view types in Pounamu -----	103
Figure 4.6 Construction of the EML event handlers in Pounamu -----	104
Figure 4.7 Tool construction steps using Marama meta-tools -----	106
Figure 4.8 Creating the EML tool using Marama -----	108
Figure 4.9 Defining EML metamodel in Marama -----	109
Figure 4.10 Defining shapes in Marama -----	111
Figure 4.11 Defining the view type in Marama -----	112
Figure 4.2 The Tree layout in Marama_EML -----	115
Figure 4.3 Process overlays in EML -----	117
Figure 4.4 Zooming commands and zoom view -----	119
Figure 4.55 Selection zoom -----	120
Figure 4.16 Fisheye view of a Diagram in MaramaEML -----	122
Figure 4.17 EML Integrated Tool Framework -----	123
Figure 4.18 Consistency Mapping Between a BPMN view and a Form Chart View -----	124
Figure 5.1 University Enrollment System Overall Structure -----	131
Figure 5.2 Using EML Overlays to Model the Enroll in a Course Process -----	134
Figure 5.3 BPMN View – Enroll a Course -----	138
Figure 5.4 Using EML and BPMN views to model the same process -----	139
Figure 5.5 BPEL Generation and LTSA Code Validation -----	141



Figure 5.6 Zoomable View in MaramaEML -----	142
Figure 5.7 Fisheye View in MaramaEML -----	143
Figure 5.8 BPEL Deployment -----	145
Figure 5.9 Hot-deployment of a BPEL process in Apache ODE -----	146
Figure 5.10 Testing process WSDL interface using the Eclipse Web Services Explorer -----	147
Figure 6.1 Selected EML Notation Examples (before CD) -----	152
Figure 6.2 Improved Flow Notations (after CD) -----	153
Figure 6.3 Service View Diagram in Early Version of EML -----	155
Figure 6.4 Property Sheet Example for Travel Planner Service -----	156
Figure 6.5 Service Integration View in Early Version of EML -----	158
Figure 6.6 Form Based Service View in Early version of EML -----	159
Figure 6.7 Tree Layout for the overall Structure -----	159
Figure 6.8 Exception View in Early version of EML -----	162
Figure 6.9 A process Overlay on EML Tree in MaramaEML 1.0 -----	163
Figure 6.10 A BPMN Diagram in MaramaEML 1.0 -----	164
Figure 6.11 Form Chart Diagram in MaramaEML 1.0 -----	165
Figure 6.12 BPEL Code Generation in MaramaEML 1.0 -----	166
Figure 6.13 User Performance Diagram -----	175
Figure 6.14 General Quality Feedback Diagram -----	176
Figure 6.15 EML and MaramaEML Usability Rate Summary -----	176
Figure 7.1 MaramaMTE Architecture View -----	182
Figure 7.2 Web Service Composition in ViTABaL-WS -----	183

## **LIST OF TABLES**

---

Table 2.1 Selected of Other Modeling Notations -----	40
Table 2.2 Comparison of Process Modeling Techniques -----	47
Table 3.1 Different Service Status -----	57
Table 3.2 Different Operation Status -----	60
Table 6.1 Participants' IT Background Distribution -----	173
Table 6.2 Participants' BPM Background Distribution -----	173

# Chapter 1

## INTRODUCTION

---

There is no doubt that business process plays a very important role in running a business. A healthy business process is the foundation of the success of an organization. The realization of all strategic objectives has to rely on business processes to achieve. Based on the result of a recent CIO survey (Information Week 2009), “streamline or optimize business processes” is the top business priority of executives. In order to achieve process excellence, people carry out various process improvement initiatives, such as business process reengineering (BPR) and business process management (BPM), which have become the fashionable terms nowadays (Hill and Brinck et al 1994; Jin 2003).

BPM has been defined as “a structured, coherent and consistent way of understanding, documenting, modeling, analyzing, simulating, executing and continuously changing end-to-end business processes and all involved resources in light of their contribution to business success.” (Recker 2008) BPM covers the overall management of organizations by looking at the lifecycle of their business processes. It is essentially a consolidated selection of tools and methods from earlier practices such as Business Process Modeling, Process Re-Engineering, Process Innovation, Process Management and Process Integration (Box and Cabrera et al 2006; Kramer and Herrmann 2007).

No matter which process improvement initiative people want to conduct, they have to understand the business processes and perform necessary analyses to design or redesign the processes. Business process modeling enables a common understanding and analysis of a business process, while computer simulation is an effective technology to diagnose business processes, especially when complexity and scope become issues. Often, as existing processes are modeled and simulated, complex relationships and behaviors are exposed and evaluated (Jung and Cho 2005; Grundy and Hosking et al 2006). Over recent decades, business process modeling has emerged as a popular management approach in Information Technology (IT) and

Business Process Management practice. Both recent and earlier studies support this statement. Business process modeling has over the last three years continuously been identified as a top business priority and building business process capability continues to be a major challenge for senior executives in the coming years. The increasing global competition and publicized cases of outsourcing and off-shoring have stimulated demand for business process management and enticed organizations to increase their engagement in BPM initiatives. A study on the current state of business process modeling has found that 58% of their 348 respondents' organizations spent up to US\$500,000 on process modeling in 2005. Roughly 15% of the surveyed organizations spent between US\$500,000 and US\$1,000,000 in 2007, and 19% spent between US\$1,000,000 and US\$5,000,000. Moreover, 53% of respondents indicated that their organizations would be increasing process modeling and management efforts in 2009 (Recker 2008; Recker and Rosemann 2010).

The strengthened interest in business process modeling has triggered substantial academic and commercial work aiming towards advanced business process management solutions. Yet, while organizations appear to be well aware of the need for business process modeling efforts, implementation remains a challenging task. Indeed, a recent study found that many organizations still struggle with an efficient modeling approach to discovering, visualizing and documenting their business processes (Recker and Rosemann 2009; Liu and Grundy et al 2007).

In Section 1.1, we discuss the main motivation of this research. Section 1.2 introduces the research targets. The structure of the whole thesis is described in Section 1.3. A list of publications is reviewed at the end of this chapter.

## **1.1 Motivation**

Business process modelling as a way of graphically articulating at least the activities, events/states, and control flow logic that constitute a business process is seen by many as a promising solution to the challenge of process discovery and documentation. Correspondingly, process modelling has over the years risen in attractiveness and is by now a popular conceptual modelling approach. Process models are created using process modelling grammars, which specify the syntax and semantics of the graphical

elements in a process model and the rules of how to combine the elements. Due to a strengthened interest in a more disciplined approach towards Business Process Management, many organizations have been motivated to make significant investments in process modelling initiatives. This, in turn, has triggered substantial related research, especially on those visual modelling approaches that are used for process modelling. In fact, the ongoing and strengthened interest in modelling for Business Process Management has over time given rise to a wide range of visual modelling methods, and consequently, a competitive market is providing a large selection of visual products for process modelling.

Examples include Entity-Relationship models (Chen. 2002), Data Flow Diagrams, Flow Charts (Urbas and Nekarsova et al 2005), Scenarios, Use Cases, and Integration Definition for Functional and workflow Modeling (Eriksson and Penker 2000). Many types of workflow management systems have been developed to model and implement enterprise business processes (Pinci and Shapiro 1993; Paussto 2005; Leymann 2001). Their goal is to specify, enact and evolve business processes using a high-level visual modeling approach. Using workflow approaches, business processes are typically modeled as stages, tasks and links. These models are used to control the execution of software components that comprise an enterprise system. Process technology can also be used to model processes executed within systems e.g. in Enterprise Resource Planning (ERP) systems. More recently, a young but rapidly growing research field, aspect-oriented modeling (AOM) (Barra and Génova et al 2004), has been recognized as a valuable approach for dealing with crosscutting concerns at early stages of software development (Gokhale and Gray 2005). This approach is used to analyze a complex system from multiple viewpoints to identify highly abstract components. Most existing Enterprise visual modeling languages adopt box-and-line style of diagrams. These generally work well for small to medium diagrams.

Nevertheless, despite the ongoing proliferation of process modeling languages, only a few have been widely accepted by practitioner communities. Existing research has shown that visual process modeling methods differ significantly in their features and characteristics, such as, for instance, their representational capabilities, their support for expressing workflow patterns or their support for formal analysis of correctness

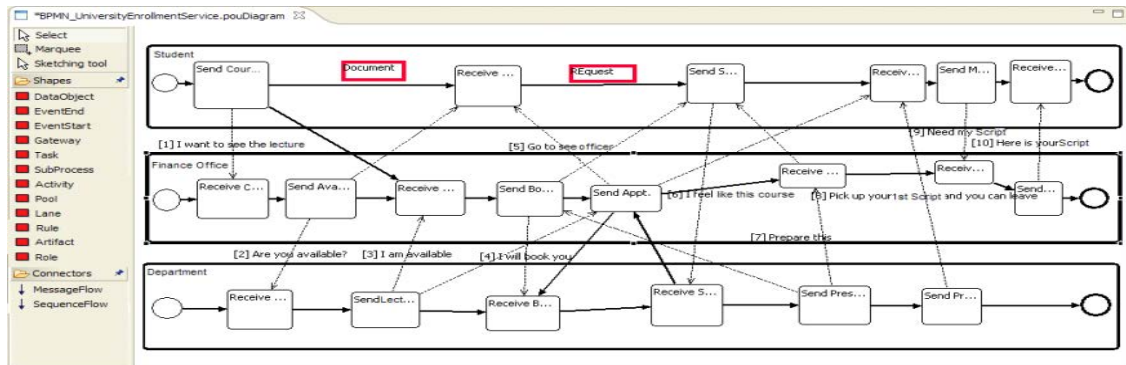
criteria (Engels and Erwig 2005; Gamma and Helm et al 1995). Actual practice, on the other hand, informs us that certain process modeling languages have achieved higher levels of adoption and dissemination in visual modeling practice than others. In fact, some available visual modeling languages exist as objects of interest only to academic scholars (Gottfried and Burnett 1997; Grundy and Hosking et al 2006).

We were asked to model a large university enrolment system as part of a process improvement exercise (Li and Hosking et al 2007). This is a complex enterprise system that involves dynamic collaborations among five distinguished parties: Student, Enrolment Office, Department, Finance Office and StudyLink (the New Zealand government's student loan agency).

The main functional requirements are:

- (1) Students will use this system to search the course database and apply for enrolment in target courses; if their application is approved, they need to apply for a loan from StudyLink;
- (2) After receiving student applications, the Enrolment Office checks the academic conditions with academic Department staff and then informs Students of the results;
- (3) Department staff check the course enrollment conditions and make the final decision (approve or reject);
- (4) For an approved enrolment application, the Finance Office tracks fee payment and informs the Enrolment Office and Department of any changes. If a Student applies for a loan, the Finance Office also needs to confirm the student information with StudyLink.
- (5) StudyLink investigates the student information with the university and then approves (or declines) the loan application.

A conventional Business Process Modelling Notation (BPMN) diagram capturing some of this enrolment process is shown in Figure 1.1. This illustrates the use of process stages, "swim lanes", process flow, etc. when modelling a process. Unfortunately as the process definition grows, the user must create either massively complex and unwieldy diagrams or "drill down" into sub-stages, introducing hidden dependencies and complex navigation (Baker 2002; Recker and Rosemann et al 2009).



**Figure 1.1: Part of a BPMN specification of the Enrolment System (Li and Hosking et al 2007)**

What we require is an enterprise modelling tool that includes a visual language that:

- a) can efficiently model distributed complex systems and related collaborations
- b) can present multi-level abstraction to assist different process specifications
- c) is easy to understand by both business and technology participants
- d) addresses the problem of modelling over-complex diagrams among distributed parties
- e) can be integrated effectively with other modelling technologies
- f) supports automatic generation from visual models to industry standard code e.g. BPEL scripts

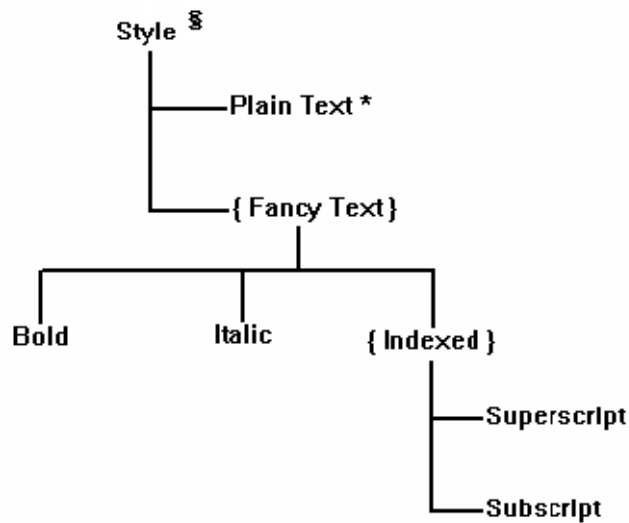
The corresponding question of the success of process modeling languages has raised high interest from us. We have conducted studies to investigate the strengths and weaknesses of specific process modeling languages (a more detailed review can be found in Chapter 2). We have evaluated various visual modelling languages and support tools to model such a system. We found that most existing modelling languages and tools only solve limited design issues. General purpose modelling languages like UML (Schnieders and Puhmann 2005) and Petri Nets (Marshall 2004) have a well-established set of modelling notations and constructs. Though they are sufficiently expressive to model business scenarios, they are difficult for a business user to learn and use (fails items c and e above). Domain specific languages like Web Transition Diagrams (WTD) and T-Web systems (Kornkamol and Tetsuya et al 2003) are very easy to understand but are limited to the scope of service level composition and modelling. They are not efficient in presenting multi-level abstractions of

business processes (fail item b). Business oriented frameworks like ARIS (Goel 2006) and TOVE (Buschmann and Rohnert 1996) are based on a generic and reusable enterprise data model technology. They also provide a holistic view of process design, but focus too much on technical processes and efficient software implementations. Hence, they can result in ambiguity of the models as extra programming knowledge is required (fails items a, c, d). Some efficient modelling languages like BPMN (BPMI 2010), BioOpera (Pautasso and Alonso 2005), Form Chart (Draheim and Weber 2005) and ZenFlow (Martinez and Patino 2005) use simple notations to represent processes and also provide support tools to automatically generate industry standard code like BPML and BPEL4WS (BPMI 2006). They all use workflow-based box and line methods to describe the system. Severe cobweb and labyrinth problems appear quickly using this type of notations to model the enrolment system (Recker and Rosemann et al 2007). Multi-view tool support has been applied in many such systems to mitigate this problem but this increases hidden dependencies and requires long term memory to retain the mental mappings between views (fails item d).

On the other hand, there are a lot of commercial tools available for business process modelling and simulation, however, despite the increasingly enhanced functionalities, there are still some obstacles in widely using these tools (Ali 2007; Baeyens 2007). The common issue is the conflict between usability and flexibility. Typically, the more flexible functionalities a tool intends to provide, the more difficult to use the tool will be (Anderson and Apperley 1990; Baker 2002).

Our earlier work (Anderson and Apperley 1990; Phillips 1995; Li and Hosking et al 2004) on modelling complex user interfaces (by Lean Cuisine+) and their behaviour with visual dialogue notations demonstrated that a tree-based overlay structure can effectively mitigate these complexity problems. Lean Cuisine+ (Phillips 1994; Li and Hosking et al 2004) is a graphical notation based on the use of tree diagrams to describe systems of menus. A menu is viewed as a set of selectable representations (called menemes) of objects, actions, states, parameters and so on, which may be logically related or constrained. It has the overlay structure for specifying the underlying behaviour of event-based direct manipulation interfaces (Phillips, 1994). Figure 1.2 shows the Lean Cuisine diagram used to describe the Style menu interface.





**Figure 1.2: Using Lean Cuisine to Present Style example (Anderson and Apperley 1990)**

Lean Cuisine+ offers a clear, concise, and compact graphical notation for describing asynchronous aspects of menu-based dialogues, but was developed explicitly for graphical user interface dialogue description. It does not have any support for business process modeling. However, the tree-overlay concept is promising to use in the modeling area. They are familiar abstractions to manage complex hierarchical data for business modellers and business people; can be easily collapsed and expanded for scalability; can be rapidly navigated; and can be over-laid by cross-cutting flows and concern representations.

Hence, the above gap motivates us to develop a novel business process modelling language and its support environment. This language adopts the tree overlay approach (from our early research in graphical user interface modelling) to mitigate the common complexity issue and cobweb/ labyrinth problems in current business modelling notations. We also aim to develop a software tool to integrate this new language with some existing notations to provide richer support for the business process modelling users.

## 1.2 Research Targets

Base on the requirements from the above real process modeling exercise and the motivation, the following research targets have been defined:

- Design of a novel modeling language (EML)

We wanted to develop a new modeling language using the tree-overlay structure to address the cobweb issues in current flow chart based modeling languages, and help to reduce the common complexity problem in the BPM domain. The language supports both organizational and process level views for the system. It should be simple and easy to understand for both business and technical users.

- Development of a software prototype for the modeling language (MaramaEML)

We developed a software tool to provide the modeling capability using EML. The tool should have the ability to integrate other modeling languages. It should allow automatic code generation of standard execution languages from visual models. It should be efficient to use for both business and technical users. The software tool also provides some extra visualization support for over-complexity of diagrams as a complementarity for EML.

- Integration of MaramaEML with other Eclipse-based tools

Eclipse is a multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system. It is written primarily in Java and can be used to develop applications in Java and, by means of various plug-ins, other languages. It is a well used open source development platform. We want our environment to have the ability to integrate with other Eclipse-based tools.

- Evaluation of EML and MaramaEML with end users

In order to increase the usability and functionality of the visual notation (EML) and its support tool (MaramaEML), we wanted to evaluate them with end users and use the direct feedback to refine our design and development. We carried out more than one evaluation during the whole development cycle.

### 1.3 Thesis Structure

Having set our motivation and targets of this research, this thesis describes the research in seven chapters. Following this introductory chapter, the remaining chapters are organized as follows:

**Chapter 2:** reviews visual modeling approaches with a focus on graphical notations. A broad range of modeling approaches, their visual notations and support environments are described in this chapter. The chapter ends with a comparison of these technologies and a summary of our main findings.

**Chapter 3:** introduces the detailed design of the Enterprise Modelling Language (EML), describing the visual representations of service tree structure, process overlay and exception handler; as well as some more advanced constructs such as dependency / trigger, iteration and condition. A case study is introduced at this point (Travel Planner System). This example is used throughout the chapter to illustrate features of the EML notation.

**Chapter 4:** describes the implementation of the MaramaEML (EML's software tool). The prototype was initially implemented using the Pounamu metatool (Zhu and Grundy et al, 2007), and then migrated to the Eclipse-based Marama (Grundy and Hosking et al, 2006) framework, and finally redeveloped using the Marama meta-tools (Li and Hosking et al 2009). MaramaEML evolves with its meta-modelling framework, and has gone through a set of improvements during the development cycles.

**Chapter 5:** presents a comprehensive case study. We use a complicated example (University Enrollment System) to demonstrate the main modeling capabilities of EML and its various support functionalities.

**Chapter 6:** discusses three different evaluations of EML and MaramaEML. We carried out different types of evaluations during the language and software

development life cycle to refine the work. The feedback analysis and improvement discussions are also described in this chapter.

**Chapter 7:** concludes the thesis. Proposals for future enhancements and extensions are also considered in this chapter.

## 1.4 Publications

While pursuing the research described in this thesis until the end of July 2010, the following refereed papers (not counting in papers in development or under review) have been published in conference proceedings.

- Li, K.N.L., Hosking, J.G., Grundy, J.C., **Li, L.** (2009): 'Visualising Event-based Information Models: Issues and Experiences', *Visual Analytics in Software Engineering, Workshop at 2009 IEEE/ACM Automated Software Engineering Conference, Proceedings of Visual Analytics in Software Engineering, Auckland, New Zealand, 16 Nov, 2009*
- **Li L.**, Hosking J.G., and Grundy J.C (2008): MaramaEML: An Integrated Multi-View Business Process Modelling Environment with Tree-Overlays, Zoomable Interfaces and Code Generation, *In Proceedings of the 23th IEEE/ACM International Conference on Automatic Software Engineering (ASE 08)*, L'Aquila, Italy (**Best Software Demo Award**)
- **Li L.**, Hosking J.G., and Grundy J.C (2007): Visual Modelling of Complex Business Processes with Trees, Overlays and Distortion-Based Displays, *In Proceedings of the 2007 IEEE Conference on Visual Languages/Human-Centric Computing (VL HCC 07)*, Coeur d'Alène, Idaho, U.S.A
- **Li L.**, Hosking J.G, and Grundy J.C (2007): EML: A Tree Overlay-based Visual Language for Business Process Modeling, *Proceeding of 9<sup>th</sup> International Conference on Enterprise Information Systems (ICEIS 2007)*, 13~19, Funchal, Madeira, Portugal
- Grundy, J.C., Hosking, J.G., **Li, L** and Liu, N (2006): Performance Engineering of Service Compositions, *Proceeding of the 2006 International Conference of Software Engineering Workshop on Service Oriented Software Engineering (ICSE-SOSE 06)*, p26 ~ p32, Shanghai, China

This research has won a significant award in 2008 at the IEEE/ACM International Conference on Automated Software Engineering, held in Italy. It is one of the top conferences in the field of Software Engineering with over 200 attendees from all over the world. The research won the ITI Tech-media Best Software Tool Demo Award for the demonstration "likely to have the most impact on industrial practice".

The two papers published in 2007 have been nominated for the “Best Research Papers” award at the Department of Computer Science, the University of Auckland.

## Chapter 2

### **LITERATURE REVIEW**

---

Since the early 1970s many languages, standards, methodologies and software tools for process modeling have been created. Most attention has been paid to the role business process modeling plays in the enterprise domain. (Li and Grundy et al 2007) The main purposes of these process modeling technologies are: enhancing the communication between stakeholders; analyzing the business requirements for future reference; generating input for enterprise design processes and helping the domain users to understand a real enterprise level world using graphical representations (Chappell 2007; Baeyens 2007).

A vast number of visual technologies have been applied in the business process modelling domain to capture graphical representations of the major processes, flows and data stores. Examples include Entity-Relationship models (Chen 2002), Data Flow Diagrams (Spönemann and Hauke et al 2009), Aspect-oriented Modelling (Gokhale and Gray 2005), Flowchart Models (Tang and Chen et al 2004), Form Chart Approaches (Draheim and Webber 2005), Scenarios (Drumea and Popescu 2004), Use Cases (Thone and Depke et al 2002), Constraint Based Languages (Vlissides and Linton 1989) and Integration Definition for Functional and Workflow Modelling (Workflow Management Coalition 1999). Despite their different visual approaches, most of these modelling technologies and their notations rely on the use of process flow or “workflow” structure to describe the business processes. In a “workflow” structure, the processes are modelled as stages, tasks and links to represent the operational aspects (Barrett and Clarke et al 1996). They focus on how systems are structured, who and how perform business tasks, what the process ordering is, how to manage information transformation, how to track the tasks, etc (Baker 2002).

Software tools help the designers to model business processes. Integrated development environments have become more and more important in the business process modeling domain. Most of the popular visual modeling languages have their own comprehensive integrated development environments. Most of these tools help business process designers reduce the amount of code that they need to produce when creating a business process diagram, and by using these tools, third party integration can be created more quickly. Software tools can also help to achieve a consistent look and feel, when different process modelers use the same process modeling tool to design their different enterprise architectures (Myers 1995a; Bederson and Meyer et al 2000).

In this chapter, we review some essential business process modeling technologies, with a focus on graphical modeling languages and their support environments. Section 2.1 presents an overview of the process modeling background. A selected range of today's enterprise level business process modeling approaches (ER Models, Data Flow Modeling, Aspect Oriented Modeling, Form Chart Approach, Integrated Modeling and Other Domain Specific Approaches), the corresponding visual languages (BPMN, UML, ER Diagram, FormChart, DataFlow Diagram, AspectM and Other Domain Specific Notations) and their support software tools are reviewed from Section 2.2 to 2.8. The business execution language will be discussed in Section 2.9. This chapter ends with a comparison of these technologies and a summary of our main findings.

## **2.1 Business Process Modelling**

Business process modelling (BPM) originally came from the manufacturing industry as a means of analysing workflows and activities in order to improve product quality and performance (Baker 2002; Ben-Shaul 1994). Today, the advancements of business process modelling have also been extended to other enterprise areas. It is a domain integrating the principles of business processes and process modelling. A business process is a collection of related, structured activities or tasks that provide one or more services for a particular or group of stakeholders. A business process can be decomposed into several sub-processes, which have their own attributes, and are performed in order to

achieve the goals of the main process. Process modelling is a method to increase the awareness and knowledge of business processes and to deconstruct the complexity of an enterprise. It is a visual approach to describe how businesses organize and perform their work (Eriksson 2000; Benatallah and Dumas et al 2003). On an enterprise level, business process modelling can be used to define information and workflows for a whole organisation and thus provide a platform for enterprise-wide management of data and processes. It integrates typical business practices, processes and information flows, data stores and system functions (IBM 2010; Berndtsson and Mellin et al 1999).

Our study shows that successful business process models in general serve two main purposes well (business and technical). On the business side, the models can be used for different levels of organizational activities. Examples include refining the scope of the project, business requirements analysis, adapting best business practices, risk management, enterprise system design, end user training, supply chain management, knowledge management and business simulation (Box and Cabrera et al 2006; BPMI 2010). On the technical side, the models can also be integrated into wider domains. Examples include groupware collaboration, process automation, software engineering, data and system integration, and transaction management (Chakravarthy and Krishnaprasad et al 1994; Chappell 2007). Most of these integrations and collaboration work rely on the conversions between graphical models and textual execution specifications.

From a detailed level, each business process is a collection of activities designed to produce a specific output for a particular stakeholder or business group. It implies a strong emphasis on how and what work has been done within the enterprise. A process is a specific ordering of work activities across time and place, with beginning, end, and clearly defined inputs and outputs (ebPML 2002; Buchmann and Bornhövd et al 2004).

From a conceptual level, a typical business process includes at least some activities, events, states, control and data flow logic. Based on those, the user also can add extra information regarding the enterprise level resources, external stakeholders, performance



Enterprise Level Business Process Modeling

Activities  
 Events  
 Control Logic  
 Metrics, communication plans etc.

Data Integration  
 Stakeholders  
 Conditions

The foundation of business process modeling is made up of four core parts: visual modeling methodologies; visual modeling notations; software tools (visual editors); and textual description languages. Figure 2.1 depicts an overview of the enterprise level business process modeling framework.

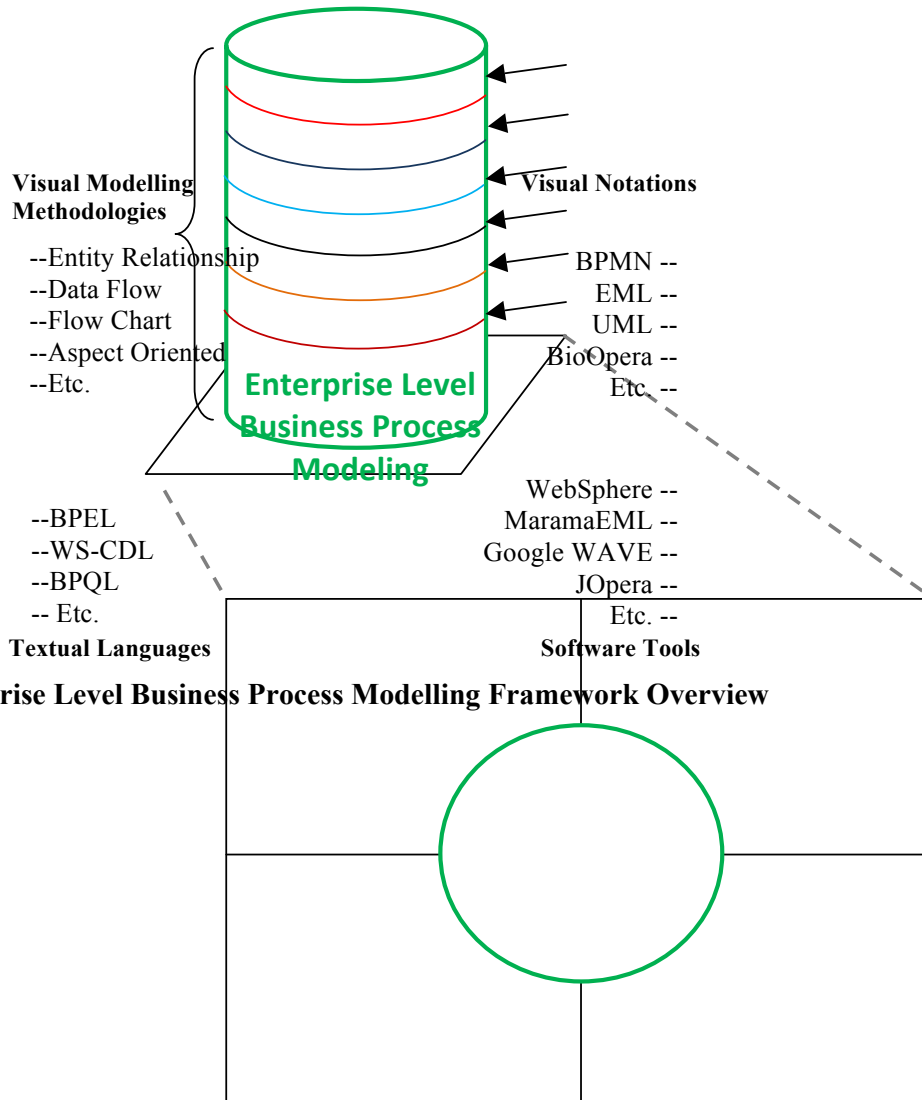


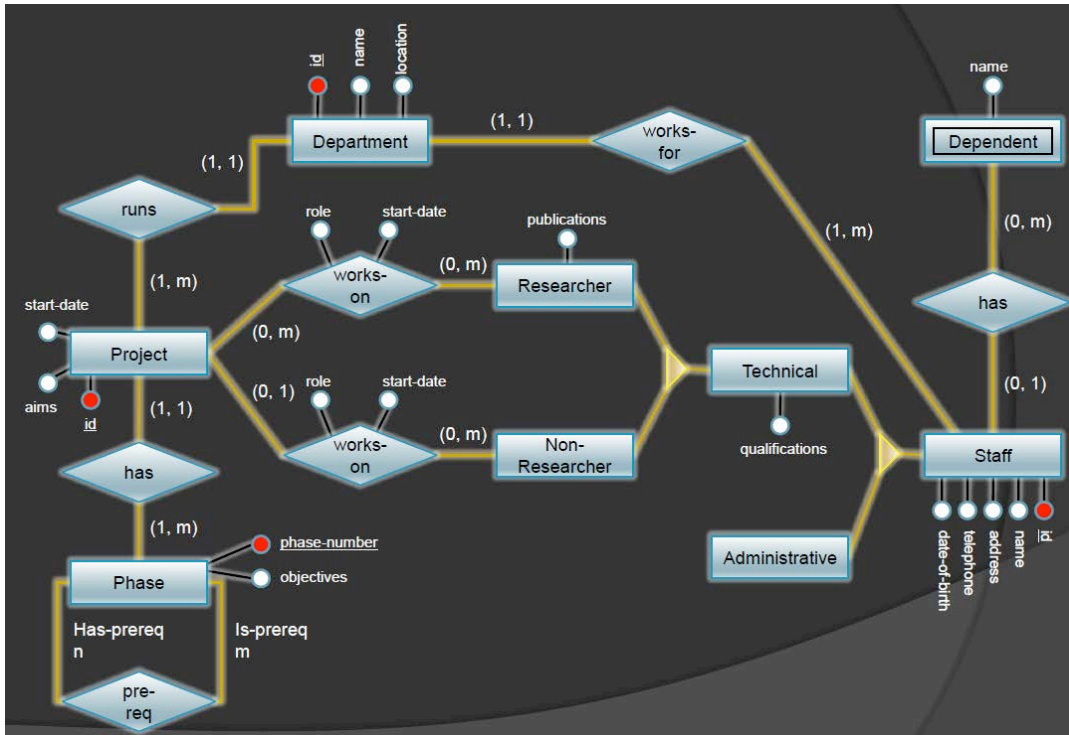
Figure 2.1: Enterprise Level Business Process Modelling Framework Overview

## **2.2 Entity-Relationship (ER) Method and ER Diagram**

ER modelling technology (Chen 2002; Hartmann and Sebastian et al 2009; Cohen 2006) describes structures of databases on a conceptual level. It is a way to graphically representing the logical relationships of entities (or objects) in order to create a database structure. The ER modeling technique can also be used to describe business processes (i.e. an overview and classifications of individual processes and their relationships).

In ER modeling, the structure for a database is portrayed as a diagram, called an entity-relationship diagram (or ER diagram). It uses a graphical approach to breakdown a system into its grammatical parts. Entities are rendered as points, polygons, circles, or ovals. Relationships are portrayed as lines connecting the points, polygons, circles, or ovals. Any ER diagram has an equivalent relational table, and any relational table has an equivalent ER diagram. ER diagramming is an invaluable aid to engineers in the design, optimization, and debugging of database programs (Chen 2002; Cox and Smedley et al 1997; Dewan and Choudhary 1991).

In the business process domain, entities are the equivalent of business nouns, such as employees, departments, products, or networks. An entity can be defined by means of its properties, called attributes. Relationships are the equivalent of verbs or associations, such as the act of purchasing, the act of repairing, being a member of a group, or being a supervisor of a department (IBM 2010; Conway and Audia et al 2000). A relationship can be defined according to the number of entities associated with it, known as the degree (BPMI 2009; Costagliola and Deufemia et al 2002).



**Figure 2.2: ER Diagram Example (Thalheim 2009)**

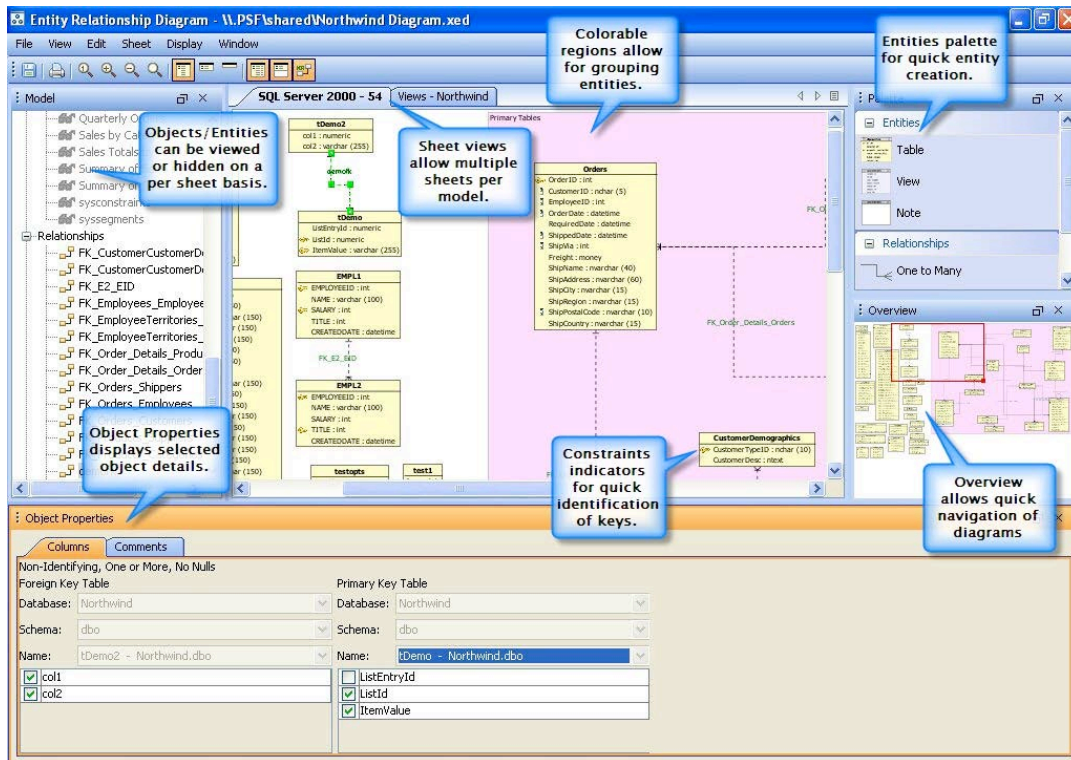
Figure 2.2 is an ER diagram example of an enterprise structure and constraints (including departments, projects, and staff). In this example, a department has administrative and technical staff. The technical staff can only be allocated to projects and those projects are always organized into phases. Phases can have other phases as pre-requisites and can also be pre-requisite for other phases. The information about the staff's dependents should also be stored, and only one single department is associated to a staff member. The technical staff can work in more than one project; however non-researchers, who are all technical staff, can participate in only one single project at a time.

The ER approach also covers data modeling quite well. In the example shown in Figure 2.2, the departments have department-id, name, and location. The staffs have id, name, address, telephone-numbers and date-of-birth. All the projects cover project-id, start-date, aims and its phases. For researchers, their list of publications and qualifications are modeled in the diagram, and for technical staff, their qualifications are represented.

There are three main components in an ER diagram:

- **Entity** - a person, object, place or event for which data is collected. The entity is represented by a rectangle and labeled with a singular noun. In the example shown in Figure 2.2, if we consider the information system for the whole organization, entities would include not only Departments, but also Project and Staff and so on.
- **Relationship** - an interaction between entities. A relationship may be represented by a diamond shape, or more simply, by the line connecting the entities. In either case, verbs are used to label the relationships. In the example above, the department runs a project, so the word "runs" defines the relationship between a department and the project(s) they run.
- **Constraint** - the data collected about the entities. The three main constraints are: one-to-one, expressed as 1:1; one-to-many, expressed as 1:M; and many-to-many, expressed as M:N.

The Entity-Relationship (ER) Modeler (Embarcadero 2010; Coupaye and Roncancio et al 1999) is a modeling tool for ER diagrams. It allows the user to create, explore, detail, and modify diagrams of the relationships and objects of a system. Changes made to an ER diagram can be automatically mapped to other associated diagrams. Manipulating the resulting diagrams will alter the relationships and objects of other corresponding diagrams. If the business uses databases, a diagram can be extracted from an existing database, and its schema objects can quickly draw for modification by the ER Diagram Generator. This tool helps to reduce the development time and improve the understanding of relationships of a process.



**Figure 2.3: Entity-Relationship Modeler Usage Example (Embarcadero 2010)**

Figure 2.3 provides a usage example of the Entity Relationship Modeler software. The model node browser at the left side presents all objects within a diagram in a tree containing all relationships, tables, views and notes. The tree can be expanded or collapsed to display more detailed information such as Indexes and Constraints. The sheet view (in the middle) shows the diagram contents in independent views for manual or automatic layout. This is the drawing area of the ER Diagram. Objects can display on more than one sheet at a time. Object properties pane (at the bottom) displays, in non-editable form, the properties of the selected object(s). The overview window (at right side) presents a bird's eye view of the current sheet, allowing fast navigation with a draggable and resizable zoom rectangle. The rectangle indicates what portion of the diagram is currently being viewed.

## 2.3 Data Flow Method and Data Flow Diagrams

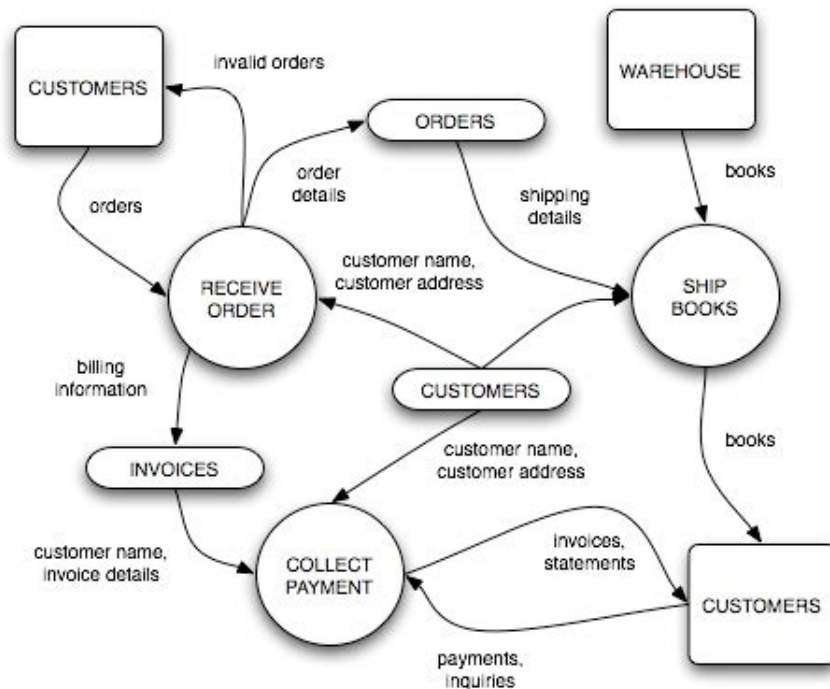
A data flow diagram (DFD) is a graphical representation of the "flow" of data through an enterprise organization (Gatzju and Dittrich 1993; Spönemann and Hauke et al 2009; Chakravarthy and Krishnaprasad et al 1994). It can also be used for visualization of processes. On a DFD, data items flow from an external data source or an internal data store to an internal data store or an external data sink, via an internal process.

In the business process domain, data flow diagrams are used to describe how the system transforms information. They define how information is processed and stored and identify how the information flows through the processes. We can use them to model the relationships among the business processes within an organization to external systems, external organizations, customers and other business processes (Grosse and Yves et al 2009; Recker 2010b; IBM 2010).

Figure 2.4 shows a data flow modeling example for a book order system. A typical data flow diagram has four main components:

- **Process** - the manipulation or work that transforms data, performing computations, making decisions (logic flow), or directing data flows based on business rules. A process receives input and generates some output. Process names (simple verbs and dataflow names, such as "Receive Order" or "Collect Payment" in the example shown in Figure 2.4) usually describe the transformation, which can be performed by people or machines. Processes can be drawn as circles or a segmented rectangle on a DFD, and include a process name and process number.
- **Store** - where a process stores data between processes for later retrieval by that same process or another one. Files and tables are considered data stores. Data store names (plural) are simple but meaningful, such as "Customers," "Orders," and "Invoices" as shown in Figure 2.4. Data stores are usually drawn as an

ellipse, rectangle or magnetic disk and labeled by the name of the data storage area it represents, though different notations do exist.

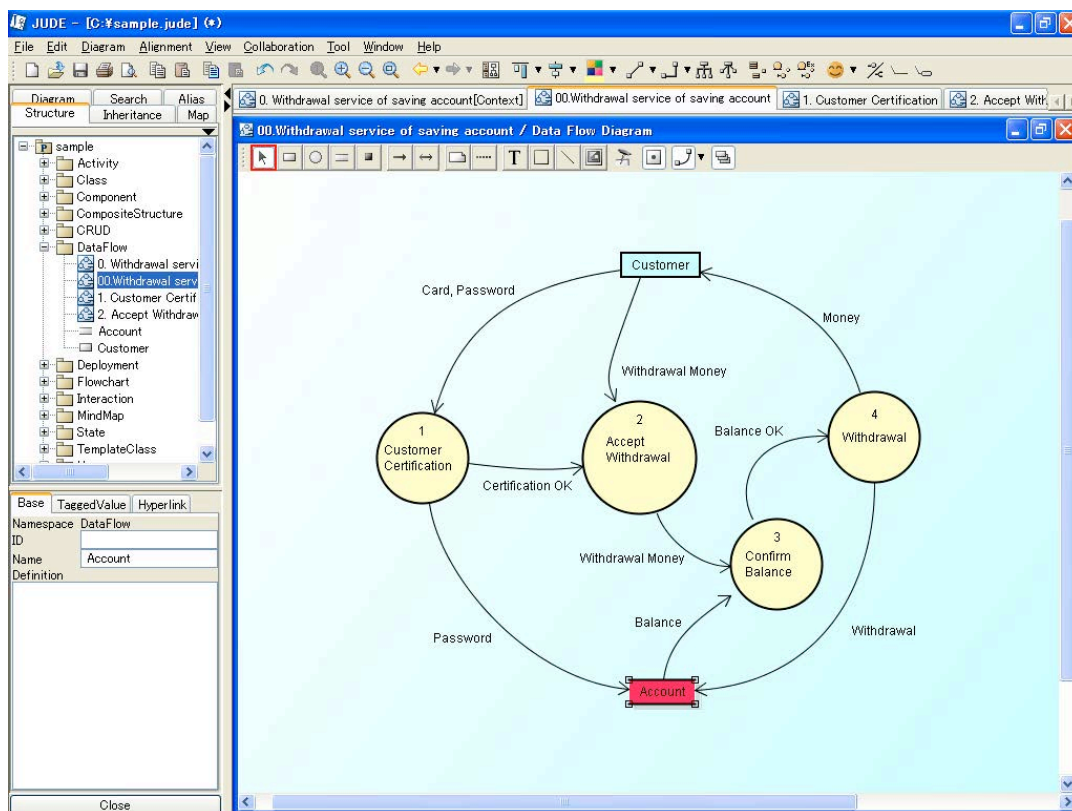


**Figure 2.4 Data Flow Diagram Example for a Book Order System (Moreira and Twan et al 2010)**

- **Flow** - the movement of data between entities, processes, and data stores. The flow portrays the interface between the components of the data flow diagram. The flow of data in a diagram is named to reflect the nature of the data used and these names should also be unique within a specific diagram (e.g. “billing information” or “customer names” in Figure 2.4). Data flow link is represented by an arrow, where the arrow is annotated with the data name.
- **Terminator** - the source or destination of data. The source in a diagram represents these entities that are outside the context of the system. They either provide data to the system (referred to as a source) or receive data from it (referred to as a sink). They are often represented as rectangles (a diagonal line

across the right-hand corner means that this terminator is represented somewhere else in diagram). Terminators are also referred to as agents, entity, or source/sink.

Figure 2.5 provides a screenshot of the JUDE Data Flow Diagram tool in use (JUDE 2010). The right side of the figure shows the main working area of this tool. Users can create their model using data flow diagram components. At the left side of the screen, a tree browser is shown. It allows the user to have an overview and navigate through a project. All elements of the system are visible and accessible through the browser. A double-click on the desired element (diagram name) brings up the appropriate drawing on the screen.



**Figure 2.5: JUDE Data Flow Diagram Tool (JUDE 2010)**

The user can use this tool to create data flow diagrams, which includes common DFD components e.g. external entity, process box, data store, data flow etc. We also can export



the hierarchical DFD data to Excel. The tool is based on a combination of the traditional data flow diagram and control flow diagram notations. It enables graphical representation of hierarchical and parallel flows and the event-driven transitions between them.

The data flow diagram (approach) focuses on only one view of a system — the function-oriented view. If we are modeling a system in which data relationships are more important than functions, an entity-relationship diagram approach will work better. Alternatively, if the time-dependent behavior of the system dominates all other issues, then a state transition diagram (approach) will be better.

## **2.4 Workflow Method and BPMN**

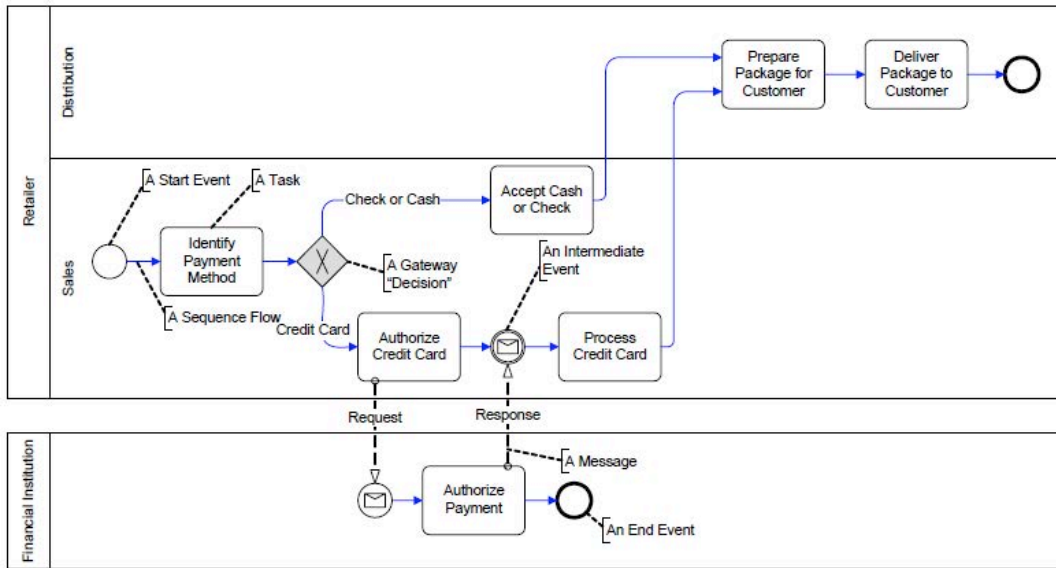
Workflow modelling (Workflow Management Coalition 1999; Zapletal and Wil et al 2009; Tang and Chen et al 2004; Felfernig and Friedrich et al 2003) is concerned with the automation of procedures where documents, information or tasks are passed between participants according to a defined set of rules to achieve, or contribute to, an overall business goal. Workflow based business process modelling is concerned with the assessment, analysis, modelling, definition and subsequent operational implementation of the core business processes of an organisation (or other business entity). Although not all business process related activities result in workflow implementations, workflow technology is often an appropriate solution as it provides separation of the business procedure logic and its IT operational support, enabling subsequent changes to be incorporated into the procedural rules defining the business process. Conversely, not all workflow implementations necessarily form part of a business process exercise, such as implementations to automate an existing business procedure.

BPMN (BPMI 2010; OMG 2009; Recker 2010) is a recently proposed workflow based process modeling language, the development of which has been based on the revision of other grammars including UML (Peltonen 2000), IDEF (Shin and Chankwon et al 2005), ebXML (Bex and Wouter et al 2010), RosettaNet (Dogac and Yusuf et al 2002), LOVeM (Gamma and Helm et al 1995) and EPCs (Dehnert 2003). The development of BPMN

stemmed from a general demand for more standardization in the area of business process management and sought to satisfy the demands related to the graphical description of business processes.

BPMN was originally incepted as a graphical grammar to complement the BPEL (IBM 2009) standard. This is the primary reason the BPMN specification contains details about the mapping capabilities between BPMN and BPEL. Due to the proposed mapping capabilities of BPMN to BPEL, the grammar has a technical focus. However, it has been the intention of the BPMN designers to develop a modelling grammar that can be applied for typical business modelling activities as well. This is why the specification document differentiates the BPMN constructs into a set of core graphical elements and an extended specialized set. The motivation behind this differentiation was to provide an intuitive basic notation that could be used to depict the essence of business processes in very easy terms whilst at the same time yielding the capacity to support complex process scenarios and formal requirements.

The complete BPMN specification defines thirty-eight distinct grammar constructs plus attributes, grouped into four basic categories of elements, *viz.*, *Flow Objects*, *Connecting Objects*, *Swimlanes* and *Artefacts*. Flow Objects, such as events, activities and gateways, are the most basic elements used to create Business Process Diagrams (BPDs) (Effinger and Johannes 2010). Connecting Objects are used to inter-connect Flow Objects through different types of arrows. Swimlanes are used to group activities into separate categories for different functional capabilities or responsibilities (*e.g.*, different roles or organisational departments). Artefacts may be added to a diagram where deemed appropriate in order to display further related information such as processed data or other comments. Figure 2.6 provides an example of a BPMN diagram. It shows a simple payment process in which customers can pay an invoice by cash, cheque or credit card.



**Figure 2.6 BPMN Example Diagram (BPMI 2010)**

After its official release in 2004, BPMN was put forward as a standard proposal to the Object Management Group and its ratification as an official standard was carried out during 2006 and 2007. Led by these standardization efforts, BPMN has encountered significant momentum in popularity and dissemination, as indicated by the growing numbers of related tool and service providers as well as of organizations that have already adapted their process modeling environments to incorporate BPMN.

Figure 2.7 is a screenshot of the WebSphere (IBM 2010) Business Modeler for BPMN. It provides functions for business process analysis as well as modeling tool capabilities BPMN. By using the software, users are allowed to make informed decisions before deployment through advanced simulation capabilities based on modeled and actual data. The system also provides code generation capabilities for languages such as business process execution language (BPEL) (IBM 2009), Web Services Description Language (WSDL) (W3C 2001) files and XML Schema Definitions (XSDs) (Bex and Wim et al 2005).

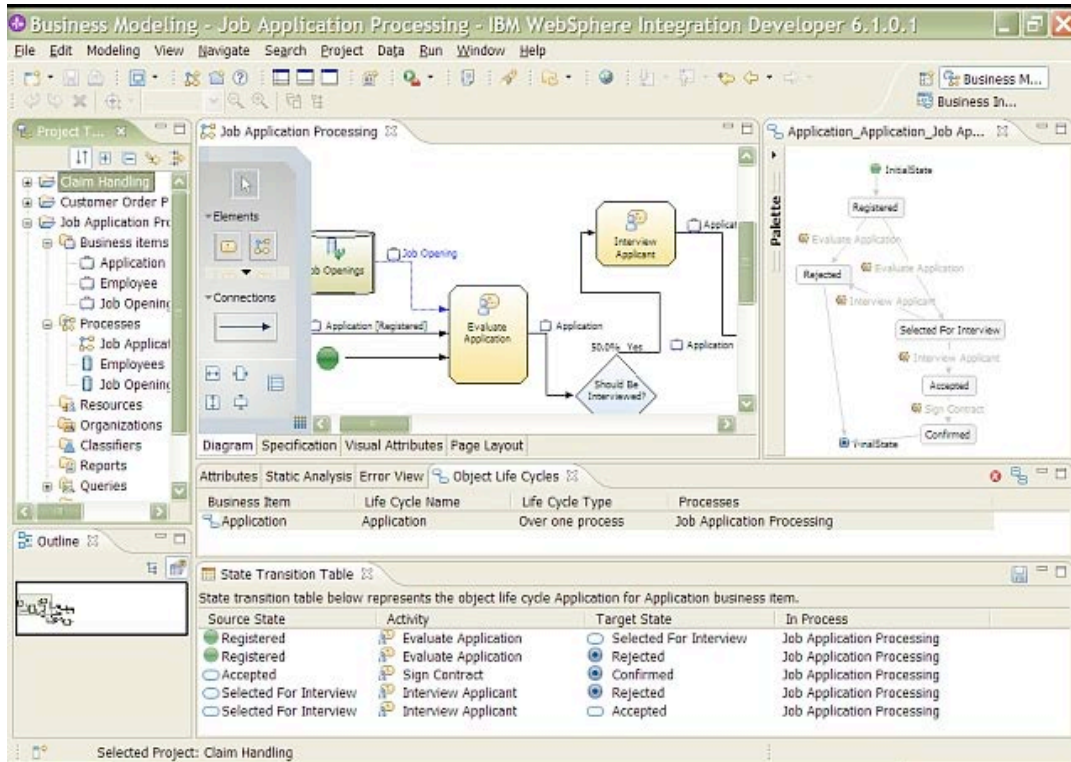


Figure 2.7: WebSphere Business Modeler (IBM 2010)

## 2.5 Integrated Modelling Method and UML

The Unified Modelling Language (UML) is a general-purpose visual modelling language that is used to specify, visualize, construct, and document the artifacts of a system (Thone and Depke et al 2002; Barra and Génova et al 2004; Foster and Uchitel et al 2003). It uses an integrated modeling method, which combines four modeling techniques: data modeling, workflow modeling, object modeling, and component modeling. So it can be used with most of the processes, e.g. business processes, logical components, process activities, programming language statements, database schemas and reusable software components etc.. It has nine different diagrams to model a system. They represent multiple views of a system. These diagrams are:

- **Use case diagram:** The use case diagram is used to identify the primary elements and processes that form the system. The primary elements are termed as "actors" and the processes are called "use cases." The use case diagram shows which actors interact with which use case.
- **Class diagram:** The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram may be capable of providing certain functionalities. These functionalities are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely characterize the class.
- **Object diagram:** The object diagram is a special kind of class diagram. An object is an instance of a class. This essentially means that an object represents the state of a class at a given point of time while the system is running. The object diagram captures the state of different classes in the system and their relationships or associations at a given point of time.
- **State diagram:** A state diagram, as the name suggests, represents the different states that objects in the system undergo during their life cycle. Objects in the system change states in response to events. In addition to this, a state diagram also captures the transition of the object's state from an initial state to a final state in response to events affecting the system.
- **Activity diagram:** The process flows in the system are captured in the activity diagram. Similar to a state diagram, an activity diagram also consists of activities, actions, transitions, initial and final states, and guard conditions.
- **Sequence diagram:** A sequence diagram represents the interaction between different objects in the system. The important aspect of a sequence diagram is that

it is time-ordered. This means that the exact sequence of the interactions between the objects is represented step by step. Different objects in the sequence diagram interact with each other by passing "messages".

- **Collaboration diagram:** A collaboration diagram groups together the interactions between different objects. The interactions are accompanied with numbers in order to help to trace the sequence of the interactions. The collaboration diagram helps to identify all the possible interactions that each object has with others.
- **Component diagram:** The component diagram represents the high-level parts that make up the system. This diagram depicts, at a high level, what components form part of the system and how they are interrelated. A component diagram depicts the components culled after the system has undergone the development or construction phase.
- **Deployment diagram:** The deployment diagram captures the configuration of the runtime elements of the application. This diagram is by far the most useful when a system is built and ready to be deployed.

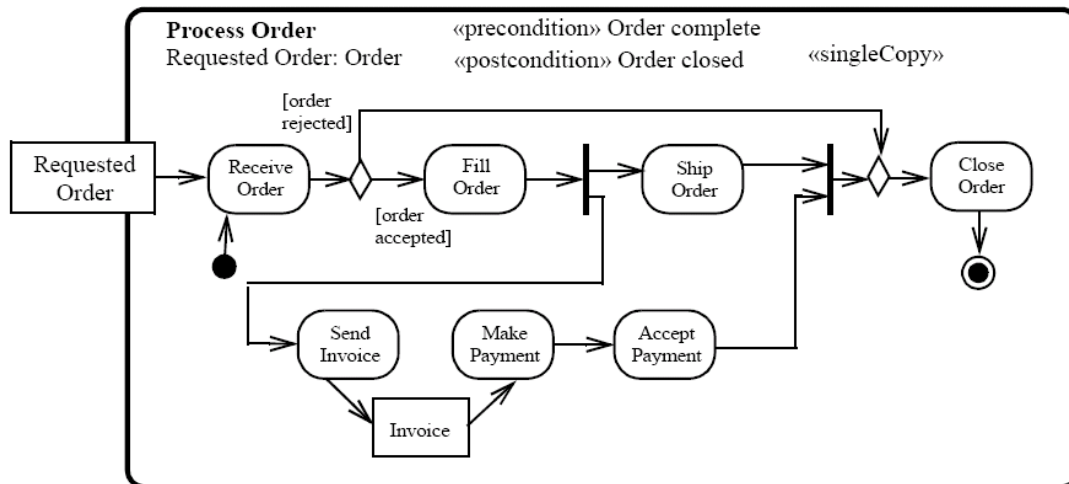
Since version 2.0, UML has provided a rich set of behavioral models which are very useful in modeling the processes, activities, people and information critical to every business (Barra and Génova et al 2004; Gugola and Nitto et al 2001). The main new features include:

- Sequence diagram constructs and notation based largely on the ITU Message Sequence Chart standard, adapted to make it more object-oriented.
- Decoupling of activity modelling concepts from state machines and the use of notation popular in the business modelling community.

- Unification of activity modelling with the action modelling added in UML version 2.0, to provide a more complete procedural model.
- Contextual modelling constructs for the internal composition of classes and collaborations. These constructs permit both loose and strict encapsulation and the wiring of internal structures from smaller parts.
- Repositioning of components as design constructs and artifacts as physical entities that are deployed.

When we use UML to model an enterprise level business process, we can divide the whole process into two parts: a structural, "static" part and a behavioral, "dynamic" part. Generally, only seven diagrams from the UML family will be used to model a business process.

- **Static Part:** describes the structural aspects of the enterprise system. The static part defines what parts the enterprise system and a business process are made up of. It includes use case diagrams and class diagrams
- **Dynamic Part:** describes the behavioral features of a system; for example, the ways a system behaves in response to certain events or actions are the dynamic characteristics of a system. It includes object diagrams, state diagrams, activity diagrams, sequence diagrams and collaboration diagrams. Figure 2.8 shows an activity diagram example for the “Product Ordering” business process.

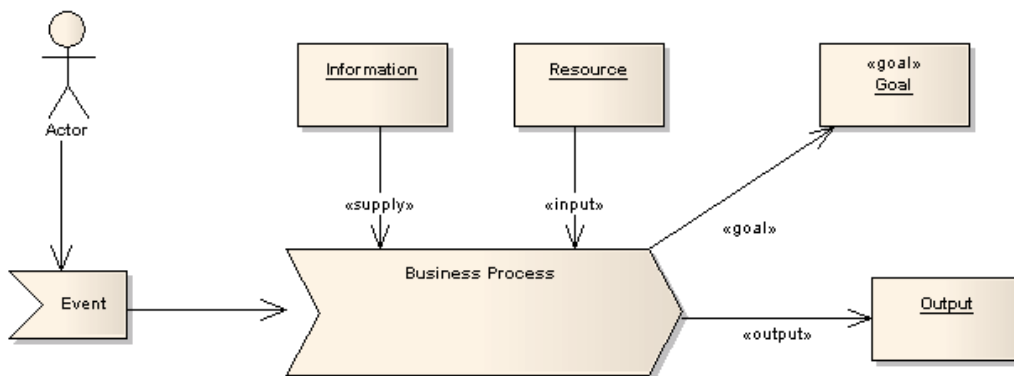


**Figure 2.8: Activity Diagram to Model Process Order**  
**(Schnieders and Puhlmann 2005)**

Instead of using the traditional UML diagrams structure to model business processes, the Eriksson-Penker Business Extension provides an alternative way to model business process using UML concepts (Eriksson and Penker 2000). It uses the notations (instead of diagrams) from the UML library, and makes some extensions in the business process modeling area. Figure 2.9 shows the general structure of this approach. In this structure:

- Supply link from object *Information*. A supply link indicates that the information or object linked to the process is not used up in the processing phase. For example, order templates may be used over and over again to provide new orders of a certain style – the templates are not altered or exhausted as part of this activity.
- Input link from object *Resource*. An input link indicates that the attached object or resource is consumed in the processing procedure. For example, as customer orders are processed they are completed and signed off, and typically a unique resource (order) is only used once.





**Figure 2.9: Eriksson Penker Business Extension Structure**  
**(Eriksson and Penker 2000)**

- Goal link to object *Goal*. A goal link indicates the attached object to the business process describes the goal of the process. A goal is the business justification for performing the activity.
- Object flow link to object *Output*. An output of one business process may feed into another process, either as a requested item or a trigger to initiate new activities.
- Object flow link from event *Event*. An object flow link indicates some object is passed into a business process. It captures the passing of control to another entity or process, with the implied passing of state or information from activity to activity.
- Goal link to Process. A Goal link indicates the attached object to the business process describes the goal of the process. A goal is the business justification for performing the activity.

Figure 2.10 represents a “Sell Books example” using the Eriksson Penker approach.



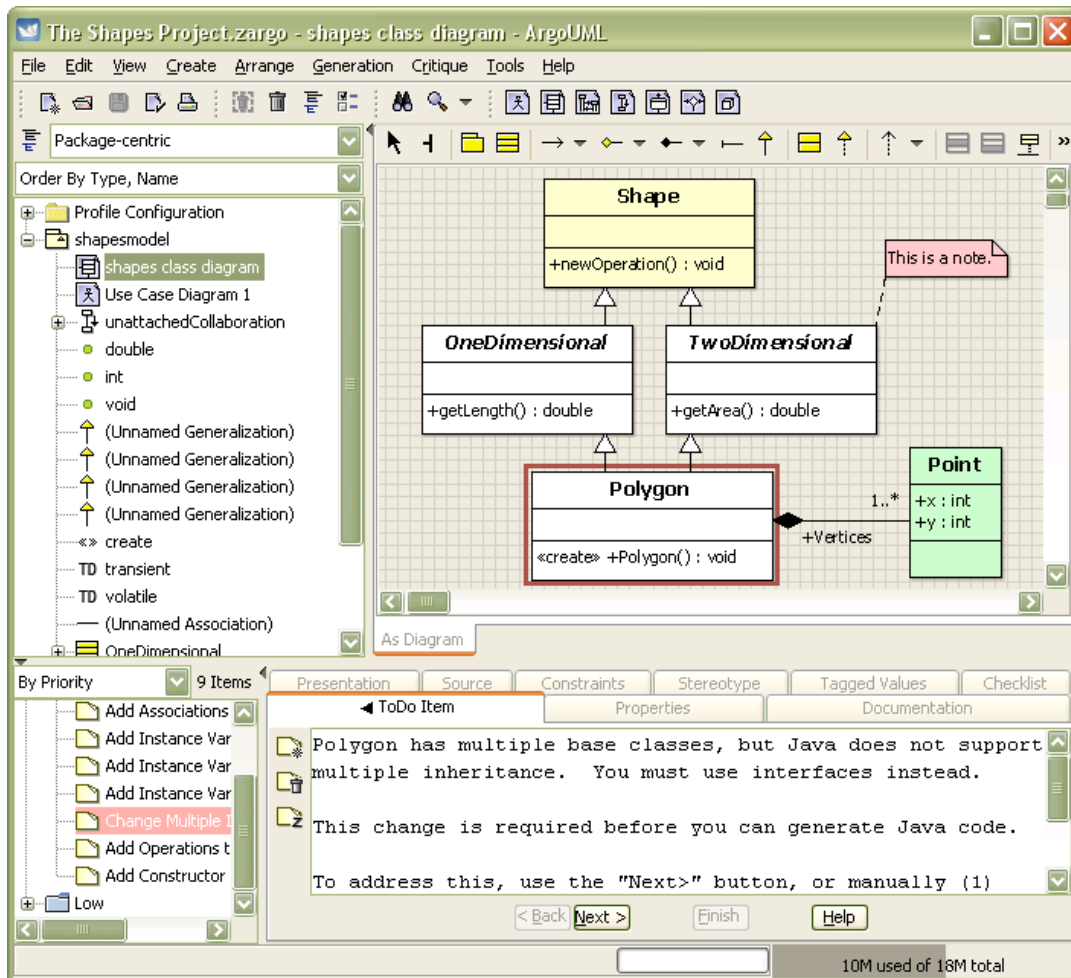


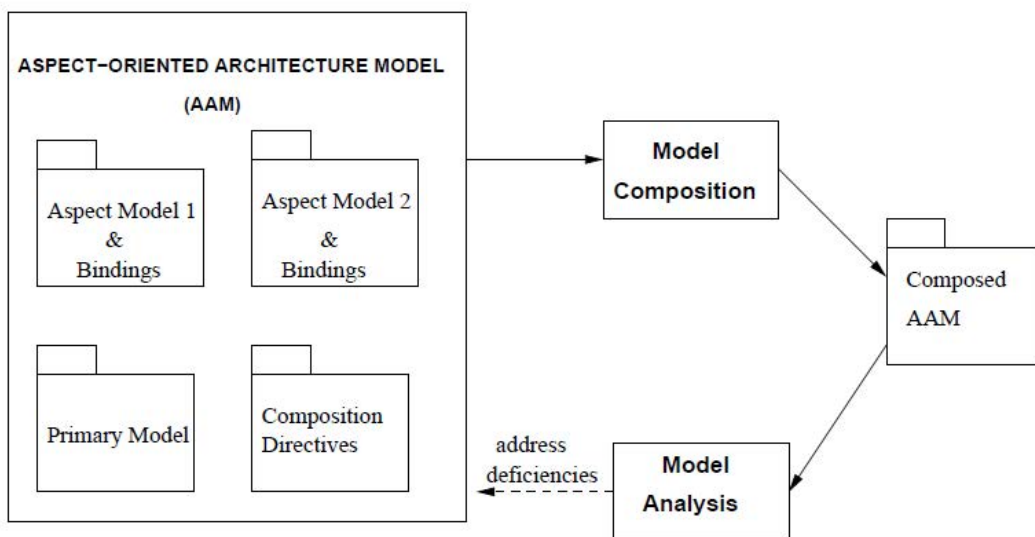
Figure 2.11: ArgoUML Software Tool (Tigris 2010)

## 2.6 Aspect-Oriented Method and AspectM

Aspect-Oriented Programming (AOP) provides a strategy for dealing with emergent entities that crosscut modularity (Barra and Génova et al 2004; Grundy and Hosking et al 2006; Ballal and Michael 2009; Kienzle and Wisam et al 2009). AOP recognizes that crosscuts are inherent in most systems and are generally not random. “Crosscut” is a common frame that two or more modelling components can connect with each other and provide their contribution. The goal of AOP is to provide new language constructs that allow a better separation of concerns for these aspects. An aspect is a piece of code that describes a recurring property of a program that crosscuts the

software application (i.e., aspects capture crosscutting concerns). AOP supports the programmer in cleanly separating components and aspects from each other by providing mechanisms that make it possible to abstract and compose them to produce an overall system.

Traditional business process modelling notations define modularization elements, such as process and activities, but have less interest to crosscutting concerns. When these concerns are mixed at several places of a same process or at different processes of a given model, it raises the complexity of the model. In contrast, aspect oriented approach for business process modelling models crosscutting concerns, coding concerns that are not localized within modular boundaries.



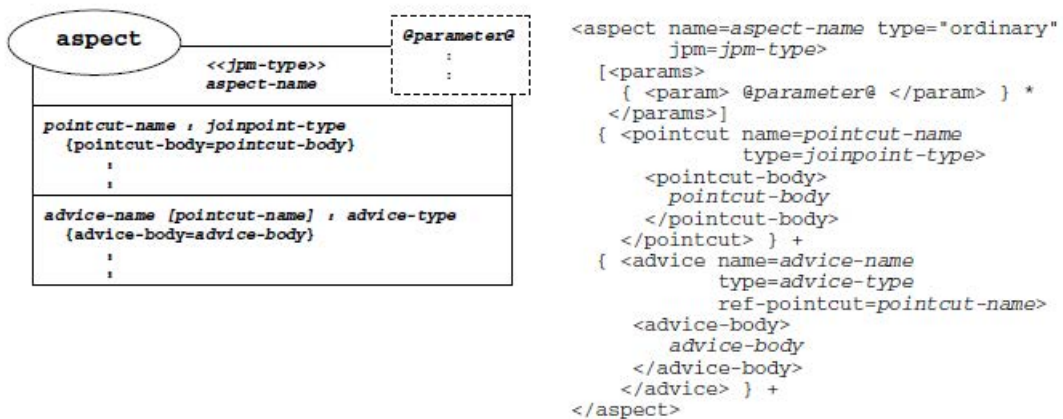
**Figure 2.12: Major Components of Aspect Oriented Modeling (Kienzle and Wisam et al 2009)**

Aspect Oriented Modelling (Kienzle and Wisam et al 2009; Ballal and Michael 2009; Meier and Cahill 2002; Hanson 2005) allows developers to define additional dimensions of separation based on system-specific concerns. In an AOM approach, aspects localize concern solutions that crosscut views described by different diagrams in a system model. The separation of crosscutting elements is a characteristic that is common to Aspect

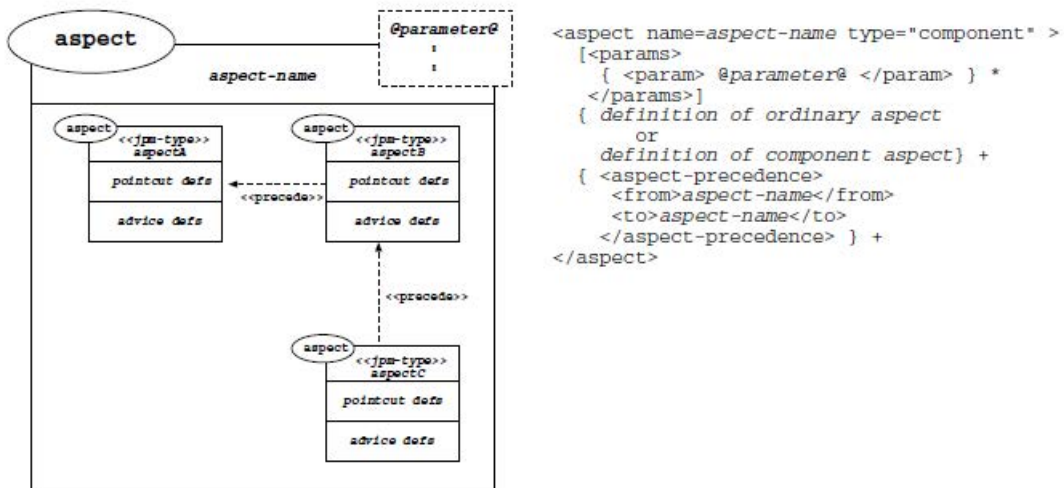
Oriented Programming and Aspect Oriented Modelling, but differences between the artifacts (models versus code) can give rise to differences in techniques. For example, at the code level there is a single representation of functionality (the source code), while a model can describe a system from multiple views using different diagrams. The views can be non-orthogonal, for example, a UML sequence diagram that describes how a set of class instances interact to accomplish a task crosscuts the class diagram view of a system. In the Aspect Orient Modelling approach, aspects describe solutions that crosscut UML models.

Figure 2.12 shows the major components of an aspect oriented modelling approach. The aspect oriented architecture model of a system consists of a primary model, aspect models and the bindings used to instantiate them in the application context, and composition directives that determine how the instantiated aspect models are composed with the primary model to produce a composed architecture model. It presents logical views of the system architecture. The Model Analysis component in this figure is responsible for analyzing the composed model to identify errors and to determine the extent that dependability objectives are met.

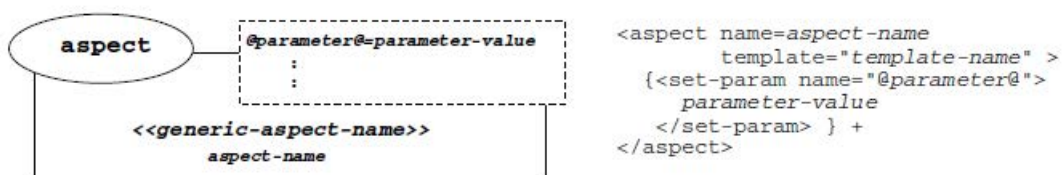
AspectM (Dantas and Walker et al 2008) is an extensible aspect oriented modelling language that provides a mechanism called metamodel access protocol for allowing a modeller to extend the metamodel and define a new join point mechanisms. A modeller can easily construct domain-specific AOM languages. In AspectM, an aspect can be described in either a diagram or an XML (eXtensible Markup Language) format. AspectM is defined as an extension of the UML metamodel. Figure 2.13 shows the AspectM diagram notations and the corresponding XML formats. Generally, the syntax of AspectM has two aspects: an ordinary aspect and a component aspect. A component aspect is a special aspect used for composing aspects. However, we can use simply the term aspect when we need not to distinguish between an ordinary aspect and a component aspect. An aspect can have parameters for supporting generic facilities. By filling parameters, an aspect for a specific purpose is generated. Using these kinds of aspects, a set of transformation steps can be described as a generic software component.



a) ordinary aspect



b) component aspect



c) parameter setting

Figure 2.13: AspectM Notations and XML forms (Dantas and Walker et al 2008)

Figure 2.14 is a software tool for Aspect Oriented Modeling. It is an open source MDSO framework implemented in Java and integrates a number of tool components. This tool supports arbitrary import model formats, metamodels, and output code formats. oAW is integrated into Eclipse and provides various plugins that support model-driven development. It contributes to and reuses components from the Eclipse Modeling Project.

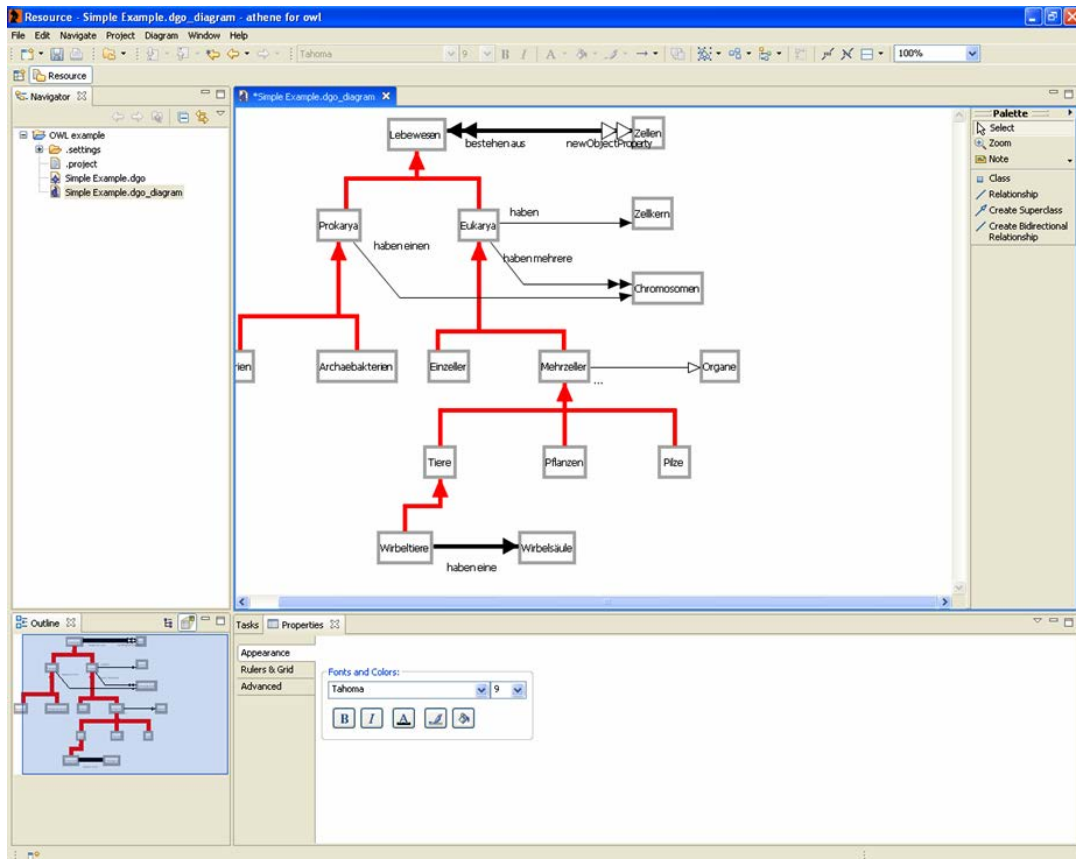


Figure 2.14: Athene for Aspect Oriented Modeling (Dantas and Walker et al 2008)

## 2.7 Form -Oriented Method and Form Chart Diagram

The FormOriented method is a technique for business systems (Draheim and Webber 2005). It defines the semantic class of systems called submit/response style applications, under which typical enterprise systems and web applications can be subsumed. Applications in this class are characterized by their type of user interface. The user of a submit/response style application fills out an electronic form, submits it to the system and

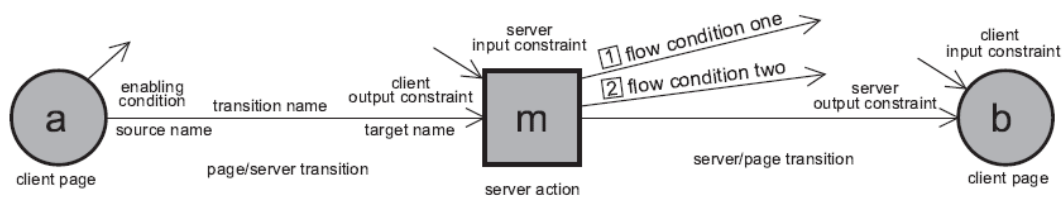
receives a response page with data and new forms. The user then submits data, partly under usage of the previously received data and so forth.

The form-oriented approach models such a submit/response style application as a bipartite state machine, which alternates between presenting a page to the user and processing the data submitted by the user. This bipartite state machine is depicted in the key artefact of Form-Oriented Analysis, the form chart.

A form chart is a visual notation for the form-oriented approach. In this notation, the user interface is set in relation to an analysis model consisting of persistent and session data. The connection is established through a data dictionary. Pages can offer collections of objects and the user can select objects from the collection and pass them back. No primary keys are passed across the state transition dataflow. Rich annotation of the state transition diagram is represented by dialogue constraints. It is given in the dialogue constraint language DCL (Suzuki 1992). In the Form-Oriented approach the responsibility for the system functionality is not artificially delegated to participating objects. Instead system functionality is modelled in a procedural style and technically delegated to the class representing the parameter list.

Figure 2.15 shows the form chart basic notational elements. The user interaction with the system, called dialogue in the following, is a sequence of interchanging client states and server states. A client state presents information to the user and offers several capabilities of entering and submitting data. The client state is called client page in the following. By submitting data the dialogue changes into a server state. In the server state submitted data is processed and depending on the current core system state the generation of a new client page is triggered, i.e. the server state is left automatically. Submitting data is conceptually like calling a method, the data being an actual parameter. The server state is called server action in the following. The transition to a client page is again considered the sending of a message, this time executed automatically from the server.





**Figure 2.15: Form Chart Notation Example (Draheim and Weber 2005)**

In above example, the transitions from client pages to server actions, page/server transitions for short, host two kinds of constraints, namely enabling conditions and client output constraints. An enabling condition specifies under which circumstances this transition is enabled, based on the state during the last server action. The enabling condition may depend on the current dialogue history. The data submitted from a client page is constrained by the client output constraint. Server actions host server input constraints. They are server action preconditions in an incompletely specified system; they must be transformed to other conditions. Transitions from server actions to client pages, called server/page transitions for short, host flow conditions and server output constraints. The flow conditions specify for each outgoing transition, under which condition it is actually chosen. The server output constraint determines which information is presented on the client page that follows in the sequel. The client input constraint is a constraint on the information on the client page, which is independent from the server page. The constraints in the form chart are written in OCL (OMG 2003).

Due to the simplicity of the notation, Form Chart diagram does not have its domain specific software available. Lots of drawing program can be used to create Form Chart diagrams (e.g. Microsoft Visio), but as a common shortage, these tools have no code generation function to share the underlay structure with other systems.

## 2.8 Other Domain Specific Modeling Languages

Domain-Specific Modelling (DSM) is about using Domain-Specific Languages (DSLs) with the expressive power gained from notations and abstractions aligned to a specific problem domain (Liu and Grundy et al 2007; Repenning and Sumnet 1995; Tolvanen 2006; Vlissides and Linton 1989; Grundy and Hosking et al 2006). It raises the level of abstraction to highlight the key concerns of the domain. Typically, DSM relies on graphical representations of the domain abstractions, as opposed to the textual form of a traditional DSL. Also, a program in a DSL is usually given a fixed interpretation, but a model in a DSM may have multiple interpretations (e.g., one interpretation may synthesize to C++, and a different interpretation may synthesize to a simulation engine).

A Domain-Specific *Visual* Language (DSVL) is capable of removing the designer from being tied to specific notations like the UML. In domain specific modelling using a DSVL, a design engineer describes a system by constructing a visual model using the terminology and concepts from a specific domain. Analysis can then be performed on the model, and then the model can be synthesized into an implementation (Wordsworth 1992; Smith 1990; Robbins and Medvidovic et al 1998).

A large number of domain specific modelling notations have been created to support business process modelling. However, the modelling methods are based on one or more of the above seven approaches. Table 2.1 provides a list of selected notations and their backend modelling approaches.

<b>Notation</b>	<b>Main Focus</b>	<b>Modelling Methods</b>	<b>Proponents</b>
Web Transition Diagrams (WTD)	Describe overall behavior of general Web applications or Web services	Form based interface + Data Flow	Jamroendararasame and Suzuki et al (2003);
JOpera Visual Composition Language	Visual composition of Web Services	Data Flow + Work Flow	Pautasso (2009); Pautasso and Alonso (2003);
ZenFlow	Composition and execution of web service	Work flow	Martinez and Patino (2005)
Extended Enterprise Modeling Language	Enterprise Modelling	Work flow + Multi-Layer Structure	Krishnamurthy and Rosenblum (1995)
Object Process Diagram	Object Process Modelling	Integrated Modeling	Dori (2002); Sturm and Dori (2003)
Semantic Modeling Notation	Process Modelling	Work Flow	Jung and Cho (2005)
Business Object Notation	System Modelling	Work Flow	Paige and Ostroff (1999)

**Table 2.1 Selected of Other Modeling Notations**

## **2.9 Business Process Execution Language**

Business Process Execution Language (BPEL) is a textual language for specifying business process behavior based on Web Services (IBM 2009). It is becoming the standard “execution” language that business process notations are compiled to. Business processes can be described in the following two ways:

- An executable business process models the actual behavior of a participant in a business interaction.

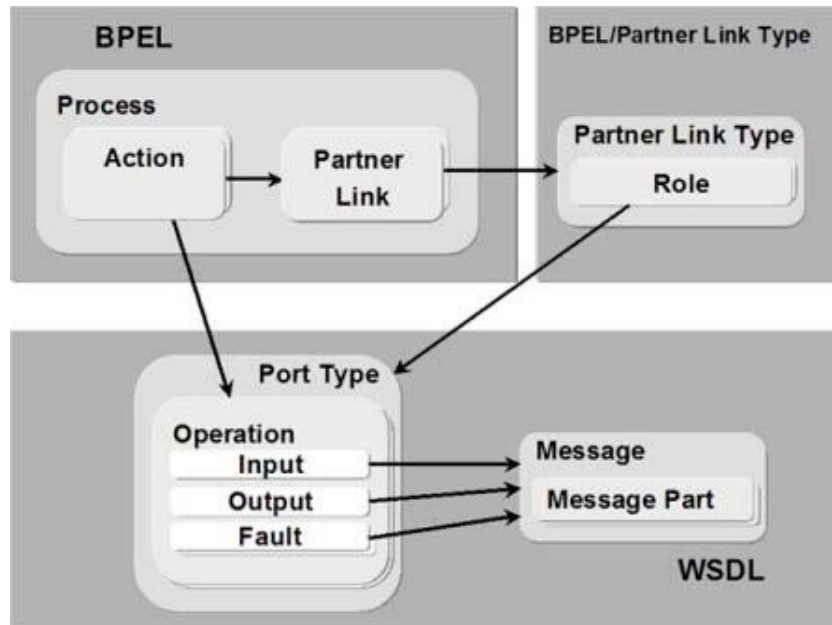
- Business protocols, in contrast, use process descriptions that specify the mutually visible message exchange behavior of each of the parties involved in the protocol, without revealing their internal behavior. The process descriptions for business protocols are called abstract processes.

BPEL is used to model the behavior of both executable and abstract processes. The scope includes:

- Sequencing of process activities, especially Web Service interactions
- Correlation of messages and process instances
- Recovery behavior in case of failures and exceptional conditions
- Bilateral Web Service based relationships between process roles

The BPEL process model is layered on top of the service model defined by WSDL. It is the notion of peer-to-peer interaction between services described in WSDL (W3C 2001). A business process defines the interaction between a process instance and its partners. To define business processes, BPEL describes a variety of XML (Bex and Wouter et al 2010) elements, such as:

- **Partners:** The actors in a business transaction
- **Containers:** The messages that need to be transmitted
- **Operations:** The type of Web services that are required
- **Port types:** The connections that are required for operations



**Figure 2.16: Mapping between BPEL and WSDL (IBM 2009)**

Figure 2.16 shows a relation mapping between the BPEL process definition and WSDL. In most cases, a BPEL program serves as a server-side service that is invoked by a client request. When a new request to a BPEL service arrives, a new instance of a BPEL program is created and further client interactions with the BPEL server are assigned to the created BPEL process until all interactions are complete. Then the BPEL process exits and disappears from the BPEL server. A BPEL business process contains two kinds of activity: a basic activity and a structured activity. A basic activity performs its intended purpose without containing further activities (e.g., assign, empty, receive, reply, invoke, etc.). However, a structured activity contains other activities (e.g. flow, sequence, if, while, etc.).

Basic activities include:

- **Do nothing <empty>** - This is the simplest basic activity in BPEL, which does nothing and so acts as an identity element in the program algebra. Sometimes it is used to consume a fault of a fault handler if there is no action for the fault. It is both a left and right identity for sequential composition.

- **Assignment <assign>** - This is another basic activity, which copies the value of an expression to a variable. In some case, a shared variable between parallel processes is protected if the assignment activity is inside a scope with attribute *isolated* = true.
- **Receive activity <receive>** - The purpose of a *receive* activity is to hold a business process waiting to receive messages from a communication channel (a partner link). A new message received from the channel will invoke the BPEL server to create a process instance of the BPEL program.
- **Reply activity <reply>** - The *reply* activity is paired with the *receive* activity; a reply activity without a corresponding *receive* activity will make the process throw a fault on reaching the end of the business process. A reply will not affect the state of a business process.
- **Invoke activity <invoke>** - The *invoke* activity is used to call a web service provided by a partner through a partner link. A partner link can be considered a channel to communicate with the web service described in the WSDL file. The type of the link is also defined in the WSDL file. As a web service can perform different kinds of operation defined in the WSDL file, the name of the operation must be declared when a service is called. An input variable is required to be passed to the service as parameter of the operation. Another variable that can be passed to the service is the output variable. It is an optional variable, if an output variable *y* is defined in the service call, the program counter will stop and wait for the answer *y* before going to the next counter. On the contrary, if the output variable of a web service call is omitted, the call is asynchronous (the program counter will go to the next counter immediately after the service is called).
- **Throw activity <throw/>** - A *throw* activity is used to throw a specific fault in an immediately enclosing scope. If a throw happens in a scope, the remaining activities of the scope are not executed and the throw will be handled by the fault

handler, if the fault handler can handle the fault; otherwise the scope is completed unsuccessfully, and the compensation handler instance of the scope will not be installed. The fault propagates to the outer scope if the fault handler of the current scope cannot catch the throw or a *rethrow* activity is executed.

Structure activities include:

- **Sequence activity <sequence>** - BPEL allows a collection of activities to be executed in sequential order through activity *sequence*. For example, a semi-colon (;) can be used to separate the different activities. These activities will be executed sequentially from the left to right.
- **Flow activity <flow>** - To improve the performance, processes are allowed to present in parallel if there is no interference between them. The parallel processes inside a *Flow* activity can be basic activities or structured activities.
- **Scope-based compensation statement** - In a BPEL scope, a compensation handler is a piece of program to undo a completed process step (scope). A compensation handler instance will be created and installed after a scope has been completed successfully. When a completed scope is to be undone, the installed instance of the compensation handler of the scope can be invoked by using *compensateScope <name of scope>*: a piece of program is executed to compensate the undoing scope and the instance of the compensation handler of the scope is uninstalled. In some cases, all completed scopes are needed to roll back; then *compensate* is used to invoke all installed compensation handler instances.
- **Compensation within repeatable constructs** - In some cases, a scope with associated compensation handler is enclosed in a repeatable construct, e.g. <while>, <repeatUntil>. The result is called a Compensation Handler Instance Group. A compensation handler instance group contains the same number of compensation handler instances as the number of successfully completed scopes

in the repeatable construct. “If an uncaught fault occurs while executing the compensation handler instance within the instance group, all running instance will be stopped and the remaining handler instances will be uninstalled” (Dehnert 2003).

There are three core BPEL components: BPEL Designer, Process Flow Template and BPEL Engine. In a typical BPEL scenario, a business expert/analyst of a company would use the BPEL Designer (a Graphical User Interface) and define the business process. A business process, for example, could be a 'Purchase Order' business scenario. Web services needed for this scenario would be included in the flow that uses the designer. Once the business expert defines the business process flow, a process logic template containing the process flow logic would be generated by the Designer in the background. At runtime, this process template would be executed by the BPEL Engine.

- **BPEL Designer** - a Graphical User Interface used to define a business process that would be independent of the underlying applications. It is intuitive for the Business experts to define the process without requiring in-depth technical knowledge. It generates the BPEL process flow logic template.
- **Process Flow Template** - adheres to the BPEL specification. It captures the business process flow logic. It is generated from the BPEL designer at design time and executed by the BPEL Engine at runtime.
- **BPEL Engine** - executes any process flow template compatible to the BPEL standard. Functionalities include the invocation of the Web services, mapping of the data content, error handling, transactionality, security, and so on. Typically, the BPEL Engine would be integrated within an Application Server.

## 2.10 Discussion and Summary



To adequately describe a business process, many forms of information must be integrated into a process model. Information that people want to extract from process models are what is going to be done, who is going to do it, when and where it will be done, how and why it will be done, and who is dependent on its being done. Process modeling languages are different in the extent to which their constructs highlight the information that answers these different questions. The differences result from the various source domains (e.g. process or software engineering etc.), as well as the visual methods used. In this chapter, we have reviewed a broad range of visual modelling methods, their modelling languages and the software tools. The analysis summary is represented in table 2.2.

	ERD	DFD	BPMN	UML	AOM	Form Chart	WTD	IOpera	ZenFlow
Process Modelling	√	√	√	√	√	√	√	√	√
Sub Process Modelling	×	×	√	√	×	×	×	√	√
Organizational Structure Modelling	×	×	√	√	×	×	×	×	×
Logical Behavioural Modelling	√	×	√	√	×	×	×	×	×
Data Encapsulation	√	√	√	√	√	√	√	√	√
Data Modelling	√	×	×	×	×	×	×	×	×
Error Handling Modelling	×	×	√	√	×	×	×	√	√
Easy to Understand and Learn	√	√	√	×	×	√	√	√	√
Functional Perspective Modelling	×	×	√	√	×	×	×	√	×
General Purpose	×	×	×	√	×	×	×	×	×
Distinguish Internal and External Events	×	×	×	×	×	×	×	×	×
DS Software Tool Support	√	√	√	√	√	×	√	√	√
Multi-View Support	×	×	×	√	×	×	×	×	×
Multi-View Integration	×	×	×	×	×	×	×	×	×
Multi Layer Structure	×	×	√	√	√	√	√	√	×
Code Generation	×	×	√	√	√	×	√	√	√
Third Party Integration	×	×	√	√	×	×	×	×	×

**Table 2.2: Comparison of Process Modeling Techniques**

## Chapter 3

# **ENTERPRISE MODELLING LANGUAGE**

This chapter introduces the syntax of the Enterprise Modelling Language (EML), describing the visual representation of service tree structure (Section 3.2), process overlay (Section 3.3) and exception handler (Section 3.4), as well as some advanced constructs such as dependency / trigger (Section 3.5), iteration (Section 3.6) and conditions (Section 3.7). A simple case study is introduced at this point (Travel Planner System), with a rather comprehensive one placed in Chapter 5. This example will be used throughout section 3.2 to 3.7 to illustrate the features of the EML notation.

### **3.1 EML Overview**

As we have discussed in Chapter 2 (Literature Review), there have been many visual languages in the area of process modelling. These kinds of languages provide a formal (or semi-formal) mechanism for the definition of business processes. A key element of such languages is that they are optimized for the operations and inter-operations of business process management systems. Such an optimization for software operations renders them less suited for direct use by humans to design, manage, and monitor business processes. Visual modeling languages have both graph and block structures, and they utilize the principles of formal mathematical models. This technical underpinning provides a foundation for business process execution to handle the complex nature of both internal and business-to-business (B2B) interactions, taking advantage of the benefits of using visual representations (Grundy and Mugridge et al 1998; Hanna 2002).

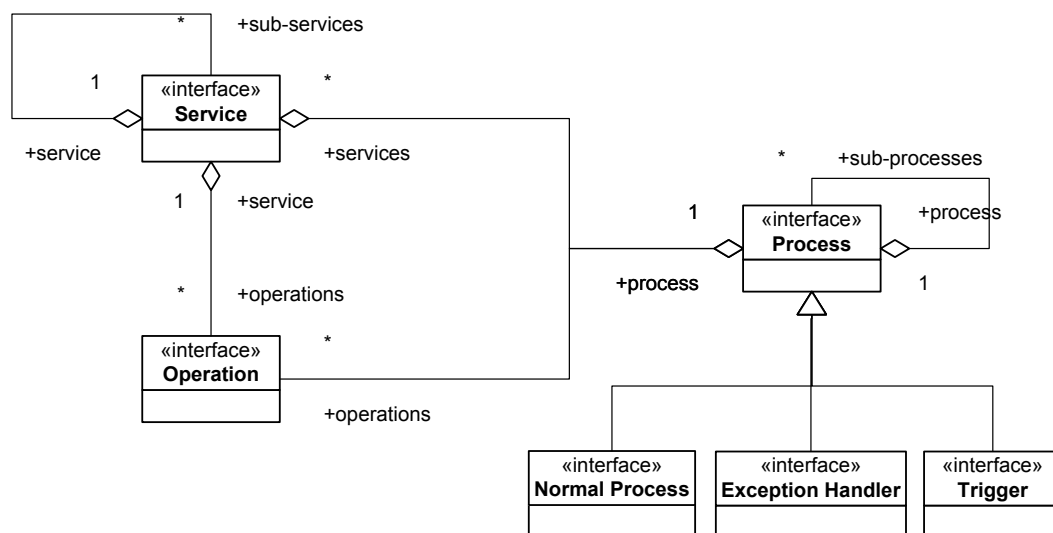
Business people are very comfortable with visualizing business processes in a flow-chart format (Draheim and Weber 2005). Most of the existing visual process modeling languages have been based on adaptations and variations of flow-chart based graphical notations and formalisms (e.g. BPMN (OMG 2009), Petri-Nets (Palanque and Bastide et al 1993), State Charts (Urbas and Nekarsova et al 2005) etc). Also within the UML community, business processes are usually modeled using Activity diagrams, for which the underlying semantics have been upgraded to Petri Nets in the current UML 2.0 proposals (Barra and Génova et al 2004). The advantage of these approaches lies in the accurate description of the workflow of a process, where a large number of constructs are devoted to describing the partial order of the services composing the process, in order to support various branching and synchronization patterns.

However, given the nature of its complexity, a complex business process could be organized in a potentially complex, disjointed, and unintuitive way that is hard to model by a single pure flow-chart based visual methodology. The deficiencies of such a visual modeling approach include:

- the lack of an efficient way to reduce the complexity and enhance the scalability of large business diagrams
- the prevalence of “cobweb” and “labyrinth” layouts (Recker and Indulska 2007) in large processes, requiring long term memory use or multi-view support which introduces many hidden dependencies
- lack of multiple levels of abstraction support
- most of them only emphasize on process modeling, missing the capability to model system functional architecture.

The language we describe in this chapter, Enterprise Modeling Language (EML), attempts to address such limitations by modeling processes primarily by a novel tree overlay structure. The principal goal that directed the design of our Enterprise Modeling Language was to provide a simple, intuitive and executable visual notation

to support rapid, user-friendly development of business processes. In this new approach, complex business architectures are represented as service trees and business processes are modeled as process overlay sequences on the service trees. By combining these two mechanisms, EML gives users a clear overview of a whole enterprise system with business processes modeled by overlays on the same view. Nevertheless, our approach does not exclude existing modeling notations. We aim to incorporate them into our EML support tool while providing additional richer, integrative views for enterprise process modeling. The objective of EML is to support business process management by both technical users and business users by providing a novel tree overlay based notation that is intuitive to business users yet able to represent complex process semantics.



**Figure 3.0: The EML meta-model**

An excerpt of the high-level EML meta-model is provided in Figure 3.0. The main meta-model entities include Service, Operation and Process. The major relationships between them include that Service is composed of Services (as sub-services) and Operations; Process involves Services and Operations and coordinates flows among them; Process may have sub-processes; and Process has three specializations in our language: Normal Process (termed simply as Process in the thesis), Exception Handler

and Trigger. The meta-model elements are mapped to their visual representations in a distinctive but consistent way. A Service is represented by a service tree; a Process is represented as an overlay cross-cutting Operations in the service tree(s); and each type of Process has its own specific overlay configuration (Process overlay, Exception Handler overlay and Trigger overlay) under a uniform representation.

## **3.2 Service Tree Structure**

Information about customer needs, technical composition of services, and service performance is fundamental to effective business process management. Service modelling is a structured approach to utilizing this information to improve the way services are delivered. Consistent application of service modelling provides the automation of processes and timely access to information. Service modelling represents a comprehensive, up-to-date overview of the enterprise system, and in the context presenting both business processes and requirements engineered from the needs to manage process resources.

We chose to represent a hierarchical decomposition of services and sub-services that make up an Enterprise application as trees. The decomposition of services into sub-parts is a standard approach in most service-oriented systems (Feng and Lee 2010; Hill and Brinck et al 1994; Haeberli 1988). We felt that a tree-based visualisation of these hierarchical decompositions would provide users with a natural way of organising related services and their sub-services.

### **3.2.1 Service / Sub-Service**

A Service is a configuration of technology designed by organizational networks to deliver to satisfy the needs, wants, or aspirations of customers (Feng and Lee 2010; Hanna 2002; Hudak 1989). In EML, a *Service* is a compound operation group that is

defined by a list of other activities (sub-services or operations). A *Sub-service* is a graphical object within a service tree, but it also can be “opened up” to show another sub-service (either *Embedded* or *Independent*). Services and sub-Services share the same shape notation in EML, a small circle. The user can pre-define different colours (as shown in Figure 3.1) to distinguish different groups of services / sub-services. All the services and sub-services also have an open centre so that pre-defined EML enhancement function icons can be included within the shape to help identify the extra functions (e.g. Elision, Reuse etc.). The name of the service / sub-service is placed outside the circle boundary and positioned arbitrarily around the notation (normally at the bottom or right side of the service / sub-service node).



**Figure 3.1: EML Service / sub-Service**

Figure 3.1 shows three different service nodes in EML. The *Travel Planner Service* node is pink in colour and its name tag appears at the bottom of the node. Two sub-services “*Customer sub-Service*” and “*Agent sub-Service*” use green and yellow colour respectively to distinguish their appearance, and their name tags are placed at the right side of the node.

In EML a service has five different execution states, they are:

















- *Un-executed / Skipped* (the default state, when the service is not invoked or is skipped)
- *Finished* (the service has completed successfully)
- *Failed* (errors were detected when the service was last executed)

- *Aborted* ( the service is killed in the middle of execution)
- *Other* (other unexpected states)

Table 3.1 lists all the service statuses in a normal service tree and when it is used with the three types of overlays. The concept of visual overlay comes from Lean Cuisine (Anderson and Apperley 1990), a graphical user interface notation. An overlay is an individual process\task or a group of related processes\tasks that is represented as a visual layer on top of the base diagram. In the Human Computer Interaction area, Lean Cuisine and its overlay technique has been proved as a good practice to represent multi-tasks structure (Anderson and Apperley 1990; Phillips 1995; Li and Phillips et al 2004). Hence we decided to adopt this approach to the business process modelling area.

The difference between the states is visually represented by the type / colour of the boundary and the status icon in the centre of the notation shape. A normal service tree (without overlays) has only the *Un-executed* status. The *Un-executed / Skipped* state shares the same style among all the three overlays. In this state, the boundary of the service uses a single line and the colour is the same as the service centre area. As to the *Finished* state, the boundary of the service is still a single line but the colour is changed to blue (in Process Overlay), green (in Exception Overlay) or red (in Trigger Overlay). The visual representation of the *Failed* state is the same across the three overlays. The fill colour is set to white and a green question mark appears in the centre of the service node. The boundary is changed to broken line for the *Aborted* state and the line colour is set specific to an overlay (blue in Process, green in Exception and red in Trigger). A star icon in the centre of the service node is used to identify the other unexpected states. Consistent colours specific to overlays are set on the icons to distinguish the overlay types.



Execution States	Service Tree	Process Overlay	Exception Overlay	Trigger Overlay
Un-executed / Skipped				
Finished	N/A			
Failed	N/A			
Aborted	N/A			
Others	N/A			

**Table 3.1: Different Service Status**

The common attributes of an EML service node include:

Attributes	Description
<b>Id</b>	This is a unique Id that identifies the service / sub-service node from other objects within the EML diagram.
<b>Documentation</b>	Textual description of the Service / sub-Service node.
<b>Name</b>	The name of the Service / sub-Service
<b>Tree</b>	A Tree must be identified for the service / sub-service to identify its location. There may be multiple trees listed if the Service / sub-Service node is a Reusable node.
<b>Parent Service</b>	If the node is a sub-Service node, then the Id of its parent service must be identified. There may be more than one parent service Id listed if this sub-Service node is a Reuse node.
<b>Child Service</b>	If the node has sub-Services, then its children's service Ids must be identified.
<b>Operations</b>	This area records the Operations directly inside this Service or sub-Service. Indirect Operations (operations belonging to this Service's sub-Services) are not included here.

<b>Service Type</b>	The service type must be Service or Sub-Service
<b>Status</b>	A Service / sub-Service's status can be Un-executed / Skipped, Finished, Failed, Aborted and Other.
<b>Input</b>	The Input attribute defines the data requirements for input to the Service / sub-Service. Zero or more Input data specifications may be defined that are required for the Service / sub-Service to be performed.
<b>Output</b>	The Output attribute defines the data format of the outputs from the Service / sub-Service. Zero or more Outputs may be defined. At the completion of the each Service / sub-Service, more than one of the Outputs may be produced. The implementation of the Service / sub-Service determines which set of data will be produced.
<b>Loop Type</b>	A service loop type can be None, Single Service Loop, Two Services Loop and More than Two Services Loop. Its default setting is None, but may be changed to others. Please refer to the Iteration section for details.
<b>Actors</b>	One or more Actors may be entered. The Actors attribute defines the human resource that will perform the Service / sub-Service. The Actors could be in the form of a specific individual, a group, or an organization.
<b>User Defined Rules</b>	The user can define Rules attributes for the Services / sub-Services. A rule is an expression that defines the relationship between services, sub-services, operations and their data. That is, if the services / sub-services are instantiated with a specified data or operation, then the appointed services / sub-services or operations must produce the specified output data or execute predefined processes. Zero (default value) or more Rules may be

	entered.
<b>Extend Properties</b>	The user can define additional properties of a service or sub-service. These Properties are “local” to the services / sub-services and are only for use within the processing of the specified service. The fully delineated names of these properties are “<Service name>.<sub-Service name>.<property name>” (e.g., “Travel Planner. Customer. User Defined Property”).
<b>Elision Type</b>	The Elision type of a Service / sub-Service must be Collapse or Expand
<b>Reuse Status</b>	This attribute indicates the Reuse status of a Service or sub-Service. It must be True (Reusable) or False (Not Reusable). The default setting is False.
<b>Reuse Id</b>	If a Service or sub-Service is reusable, this is a unique Id that identifies the node in the Reuse Library.

### 3.2.2 Operation

An Operation is an atomic activity that is included within a Service. An Operation is used when the function in the Service is broken down to a finer level of Process Model detail. Operations are the leaf nodes of the Service tree. A square shape (with orthogonal corners) represents an atomic operation inside a service (the operation and service are connected by a tree branch). The user can use different fill colours in operations to distinguish different operation groups; a light grey is used by default. The Operations also have an open centre so that EML pre-defined enhancement function icons can be included within the shape to integrate other functions (e.g. Exception Handler). The name of the Operation is placed outside the rectangle boundary positioned arbitrarily around the notation (normally at the bottom or right side of the node). A normal Operation square is drawn with a single thin black line. But in certain circumstances (e.g. Single Loop Operation, in Process overlay, in

Dependency Trigger etc.), EML changes the boundary style to represent additional information.

Figure 3.2 shows three Operation nodes and different name tag positions in EML. The *Send Book Request* node, which belongs to the *Customer Service* node in Figure 3.1, is in the default light grey colour and its name tag appears at the right side of the node. *Check Enquires* is an operation of the *Agent Service* node. It uses a dark yellow colour and its name appears at the bottom of the rectangle. The *Make Payment* operation is in red with its name at the left side of the shape. All of them have single thin black line boundary, which means they are all in the normal working status (i.e. no loops, processes or triggers are applied on these operations).



**Figure 3.2: EML Operation**

Like the *Service*, an *Operation* also has five different states as listed in Table 3.2. The colouring, bordering and fill icon conventions are identical to service nodes.

Execution States	Service Tree	Process Overlay	Exception Overlay	Trigger Overlay
Un-executed / Skipped				
Finished	N/A			
Failed	N/A			
Aborted	N/A			
Others	N/A			

**Table 3.2: Different Operation Status**

The common attributes of an EML operation node are:

<b>Attributes</b>	<b>Description</b>
<b>Id</b>	This is a unique Id that identifies the Operation node from other objects within the EML diagram.
<b>Documentation</b>	The text documentation of the Operation node.
<b>Name</b>	The name of the Operation.
<b>Tree</b>	A Tree must be identified for the Operation to identify its location. There may be multiple trees listed if the operation node belongs to a Reusable service / sub-service.
<b>Service</b>	A Service Id must be identified for the Operation to identify its location. There may be multiple services listed if the operation node belongs to a Reusable service / sub-service.
<b>Operation Type</b>	An operation type can be Receive, Send, User, Script, Abstract, Manual, Reference and None.
<b>Input</b>	The input data of the Operation (possibly none). This indicates that the data will be received at the start of the operation, after the availability of any defined Input. A corresponding outgoing Flow may be shown on the diagram. In order to reduce the complexity of the diagram, we are using the data encapsulation in EML.
<b>Output</b>	The delivery of this output marks the completion of the Operation. It can be none if required. A corresponding incoming flow may be shown on the diagram. In order to reduce the complexity of the diagram, we are using the data encapsulation in EML.
<b>Status</b>	An Operation's status can be Un-executed / Skipped, Finished, Failed, Aborted and Other.
<b>Loop Type</b>	An operation loop type can be None, Single Operation Loop,

	Two Operations Loop and More than Two Operations Loop. Its default setting is None, but may be changed to others. Please refer to the Iteration section (Section 3.6) for more detailed information.
<b>Actors</b>	One or more Actors may be entered. The Actors attribute defines the human resource that will perform the Operations. The Actors could be in the form of a specific individual, a group, or an organization.
<b>Script</b>	If the operation type is a script. It may include a script that can be run when the Operation is performed. If a script is not included, then the Operation will act equivalent to an Operation Type of None.
<b>Implementation</b>	This attribute specifies the technology that will be used by the Actors to perform the Operation. The value can be a Web Service, Application, Other or Unspecified.

### 3.2.3 Tree Layout

EML uses a tree layout to represent the basic structure of a service. We chose to use trees as they are familiar abstractions for managing complex hierarchical data for business modellers and business people; they can be easily collapsed and expanded to provide scalability; they can be rapidly navigated; and they can be over-laid by cross-cutting flows and representations of concerns. Earlier work on modelling complex user interfaces and their behaviour with tree-based overlays demonstrated these benefits (Li and Phillips et al 2004; Philips and Scogings 1998a).

All the Services, sub-Services and Operations are organized in a hierarchical based tree structure to model the system. The connection among these three components relies on the functional relationship between each other. The basic rules are:

- An Enterprise system must have at least one Service Tree.
- Every service tree must have only one Service node. It may (or may not) include an arbitrary number of sub-Service nodes.
- A Service node is always at the top of the single service tree structure. It must include at least one Operation node (directly or indirectly). It may include an arbitrary (possibly zero) number of sub-Services.
- A sub-Service is contained inside a Service or sub-Service node. It must include at least one Operation (directly or indirectly) and may have an arbitrary (possibly zero) number of sub-Services.
- An Operation is the leaf node of the service tree. It cannot include any Service, sub-Service or other Operation.

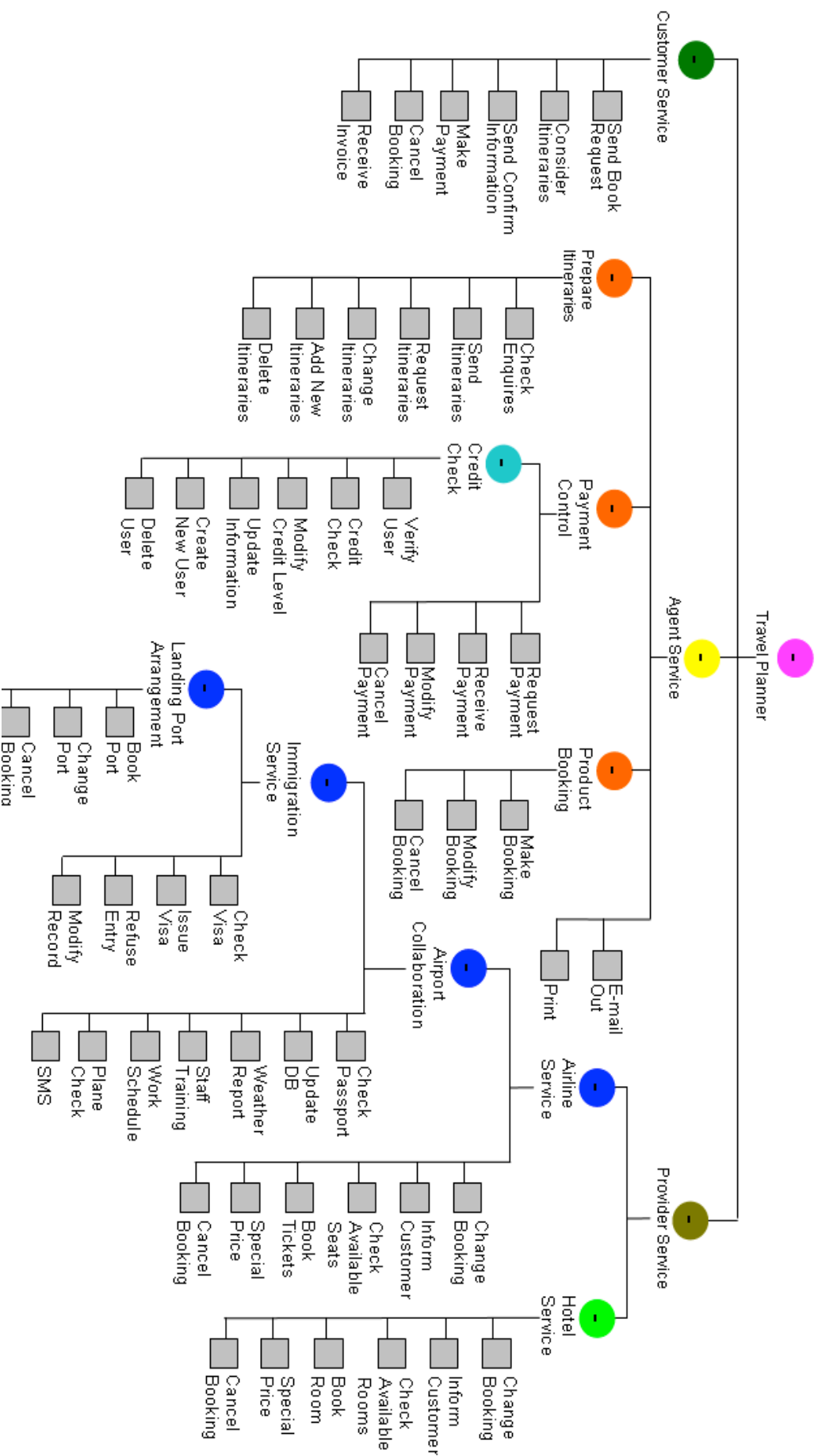




Figure 3.3 shows a complex, fully-expanded overview of an EML tree model for a Travel Planner service. A Travel Planner is a web service-based enterprise application that helps users to organize trips. Customers use the client application to submit travel enquiries to a travel agent. The agent service receives the requests and communicates with travel providers (Airline, Hotel etc.) to find out a suitable booking.

- A *Travel Planner* is a Service node in the tree. It has three sub-Services (*Customer Service*, *Agent Service* and *Provider Service*).
- There are six Operations inside *Customer* sub-Service (*Send Booking*, *Consider Itineraries*, *Send Confirm Information*, *Make Payment*, *Booking and Receive Invoice*).
- The *Agent* sub-Service includes another three sub-Services (*Itineraries*, *Payment Control* and *Product Booking*) and two Operations (*E-mail Out* and *Print*). There are also lists of different Operations inside each of the sub-Services in the above three sub-Services.
- The *Provider* sub-Service has a multi-level hierarchical sub-Service structure. It has *Airline* and *Hotel* sub-Services at the first level, *Airport Collaboration* sub-Service is embedded in *Airline* sub-Service (at level two). The *Immigration* sub-Service is inside *Airport Collaboration* sub-Service at level three. Meanwhile, it also includes another bottom-level sub-Service *Landing Port Arrangement*. There are twenty eight Operations involved in this multi-levels sub-Service tree.

The common attributes of an EML tree include:

<b>Attributes</b>	<b>Description</b>
<b>Id</b>	A unique Id that distinguishes the service Tree from others.
<b>Version</b>	The Version number of the Tree.
<b>Name</b>	A textual name for the service Tree.
<b>Creation Date</b>	The date when this version of the Tree was created.

<b>Modification Date</b>	The date on which this version of the Tree was last modified.
<b>Operation Type</b>	An operation type can be Receive, Send, User, Script, Abstract, Manual, Reference and None.
<b>Documentation</b>	The textual description of the Tree.
<b>Processes</b>	A Service Tree contains zero or more business processes. Their Process IDs are recorded here.
<b>Dependency</b>	A Service Tree contains zero or more Dependencies. Their Dependency IDs are recorded here.
<b>Exception Handler</b>	A Service Tree contains zero or more Exception Handlers. Their Exception Handler IDs are recorded here.
<b>Service</b>	A Service Tree contains only one Service. Its Service ID is recorded here.
<b>Sub-Services</b>	A Service Tree contains zero or more sub-Services. Their sub-Services IDs are recorded here.
<b>Operations</b>	A Service Tree contains one or more Operations. Their Operation IDs are recorded here.

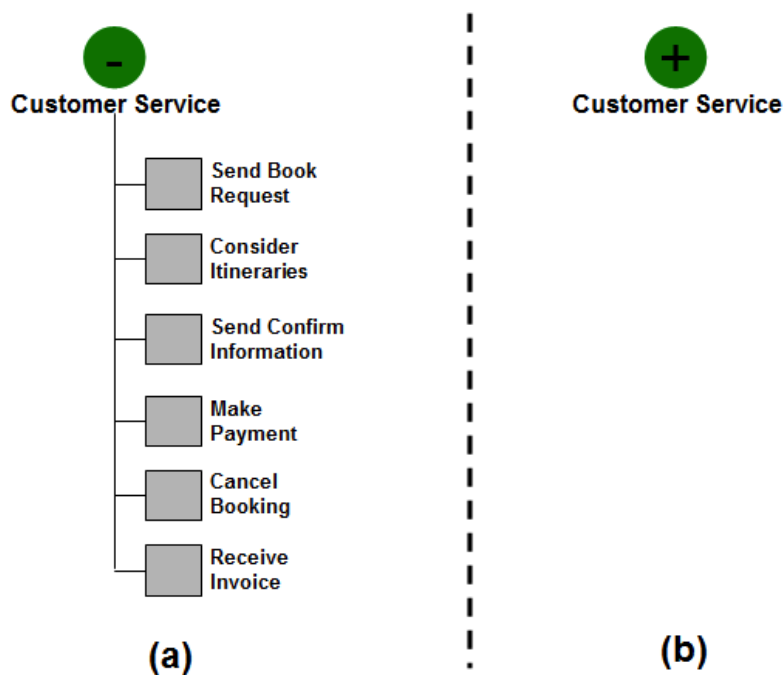
### 3.2.4 Elision

In order to mitigate the complexity of the diagram, we use symbols inside each service to identify the elision level of the service visualisation. As we adopted a tree-based visualisation of service decompositions, a collapse/expand elision mechanism is a natural way to provide complexity management. Most users are familiar with such an approach due to its commonality on graphical user interfaces and desktop user interfaces.

In EML a Service or sub-Service can be in a collapsed (elided) mode that hides its details (see Figure 3.4 (b)) or in an expanded mode that shows its details within the

view of the service in which it is contained (see Figure 3.4 (a) ). The collapsed and expanded forms of the Service / sub-Service objects are distinguished by two markers. A minus (-) symbol indicates all activities in the service have been expanded. A plus (+) symbol indicates that part or all of the sub-tasks (services and operations) are elided.

The marks are positioned at the centre of the Service / sub-Service circle shape. Every Service / sub-Service in the diagram has an elision attribute value (*Elision Type*) to provide users with the freedom to control the size of the diagram via elision of selected parts. The elision function only applies to the Services or sub-Services and cannot be used on Operations.



**Figure 3.4 (a): Extended Customer Service; (b): Collapsed Customer Service**

Figure 3.4 (a) shows an extended *Customer* sub-Service branch in the Travel Planner example. All the operations (*Send Book Request*, *Consider Itineraries*, *Send Confirm Information*, *Make Payment*, *Cancel Booking* and *Receive Invoice*) under this sub-Service are expanded for the user to get a complete view. The centre of *Customer* sub-Service node displays a minus (-) symbol. Figure 3.4 (b) presents the collapsed

situation for the same service node (all the operations belongs to this sub-Service are hidden). The plus (+) symbol appears in the centre of *Customer* sub-Service.

### 3.2.5 Service Reuse

Most service-oriented systems support some form of service packaging and reuse (BPMI 2010; IBM 2009). EML also supports service reuse to reduce structural complexity and to increase modelling efficiency. In EML we chose to represent reusable components using a separate tree. This preserves the overall approach of tree-based decompositions adopted for EML and allows one service tree to reuse elements in another, reusable service tree.

The user pre-defines its structure and saves it in a library. Reusable components have a unique ID for future usage. The user can easily attach a reusable component to any branch of an EML tree. The reusable services share the same attributes of Service / sub-Service (in Section 3.2.1). But their “*Reuse Status*” is “True” and “*Reuse ID*” is a unique number beginning with “R” (e.g. R1). If a reusable service is attached to the EML tree branch, the “*Elision Type*” attribute will be automatically set as “*Collapse*”. However, the user can change it to “*Extend*”.

The Reusable Service shares the same basic shape of the Service / sub-Service, a circle with an open centre so that Reuse ID can be placed within the circle to indicate variations of the services in the reuse library. The elision symbol’s position is moved to the bottom of the Reuse ID. The circle must be drawn with a single black thin line and the centre part must be in white. The name of the Reusable Service is placed outside the circle boundary and positioned arbitrarily around the node (normally at the bottom or right side of the service / sub-service node).

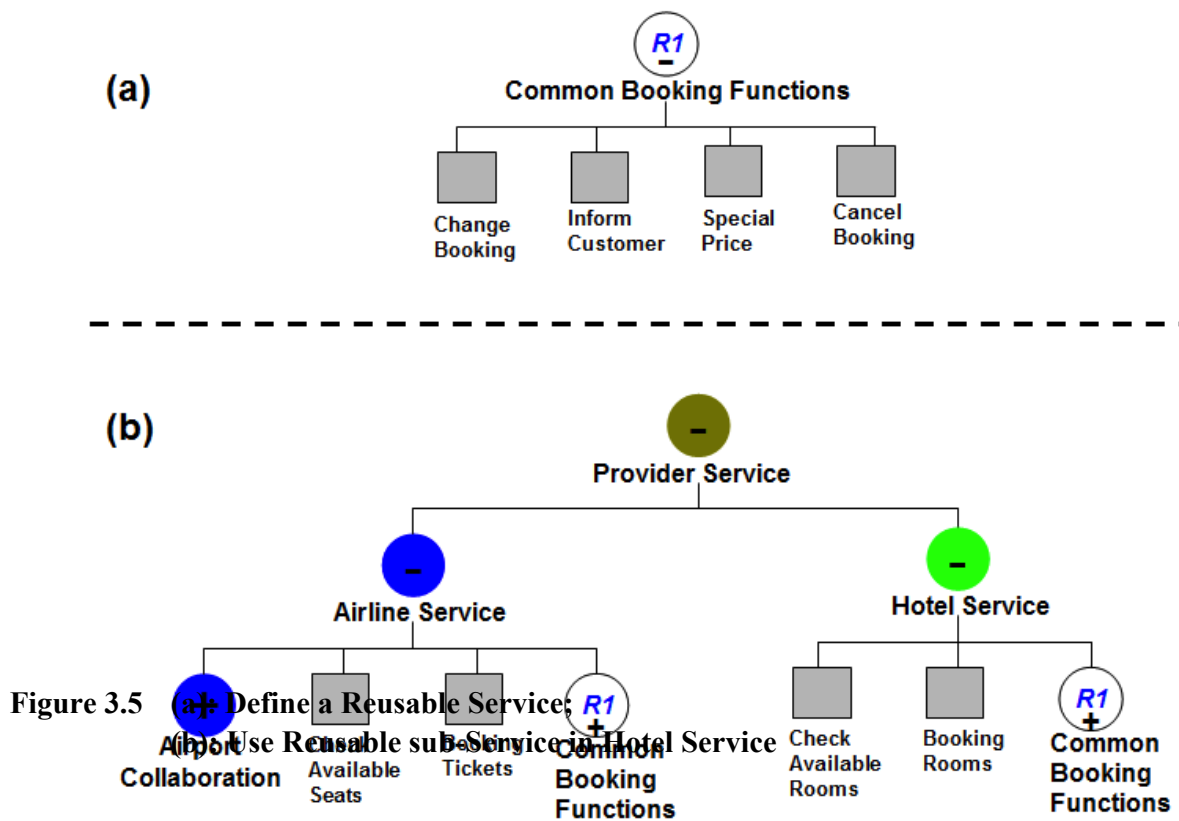


Figure 3.5 represents a Reusable Service’s definition and usage example in the Travel Planner case study. From the *Travel Planner* Service Tree example in Figure 3.3, we find that the *Airline* sub-Service and *Hotel* sub-Service have several identical operations under their tree branch (e.g. *Change Booking*, *Inform Customer*, *Special Price* and *Cancel Booking*). In order to reduce the redundancy, we combine these common operations together as a new Reusable sub-Service. Figure 3.5 (a) defines this reusable sub-service (*Common Booking Functions*). It includes four commonly used Operations and the elision type is “Extend”. The *Reuse ID* at the centre of the circle is called *R1*. Figure 3.5 (b) demonstrates the usage of this Reusable Service (*Common Booking Functions*) in the *Hotel* and *Airline* services. The user directly attaches *R1* service node to both tree branches. The elision status of this “*Common Booking Functions*” service has been changed to “Collapse” automatically.

### **3.3 Process Overlay**

A Business Process is a collection of interrelated tasks performed within a company or organization, which solves a particular issue. A fundamental part of business process modelling is the representation of flow between stages. We chose to use an approach of “overlays” to represent different kinds and instances of process flow in EML. This choice was based on the successful adoption of the overlay concept in earlier modelling efforts for the user interface domain (Philips 1993; Li and Phillips et al 2004). While EML is a very different domain of modelling, the success of tree overlays in Lean Cuisine+ leads us to adopt a similar approach for business process flow modelling in EML.

In EML each business process is represented as an overlay on the basic tree structure or an orchestration between different service trees. In a process layer, users have the choice to display a single process or collaboration of multiple processes. By modelling a business process as an overlay on the service tree, the designer is given a clear overview of both the system architecture and the process simultaneously. Processes can be elided mitigating the cobweb problem commonly existed in flow-based visual notations. In EML, a Business Process may contain more than one separate sub-Process. Each Process may have its own Sub-Processes or share (Reuse) sub-Processes with other Processes. The individual sub-Processes are independent, but could have data connection with others.

#### **3.3.1 Process Start**

The Process Start notation indicates where a particular Process will start. In EML, a Process Start icon starts the flow of the Process, and thus, will not have any incoming Process Flows. A rectangle (with orthogonal corners) represents a Process Start. The

outline and fill colours for this shape are both blue. The notations also have an open centre so that the process names and start conditions can be included within the shape. The name of the process is placed in the centre of rectangle boundary and the font is white. An annotation (double sided arrow) for start conditions may appear at the bottom of the shape.



**Figure 3.6 (a): Process Start without Conditions; (b) Process Start with Conditions**

Figure 3.6 shows two Process Start notations for the *Travel Booking Process*. In (a), the icon is a blue rectangle with only a white process name in the centre of the shape. This means the system can start the *Travel Booking Process* without any conditions. There is an additional white double sided arrow in the bottom of the Process Start shape in Figure 3.6 (b). This notation means that the *Travel Booking Process* will be started only when specified conditions are satisfied. The start condition can be a single message, certain time or a more complex set of requirements.

The common attributes of a Process Start notation include:

Attributes	Description
<b>Id</b>	A unique Id that distinguishes the Process Start notation from other notations.
<b>Name</b>	The textual name of the Process.
<b>Start Conditions</b>	The conditions to start this Process, the value of conditions can be: <ul style="list-style-type: none"> <li>• None (no icon appears in the notation): This process can be started without any conditions.</li> </ul>

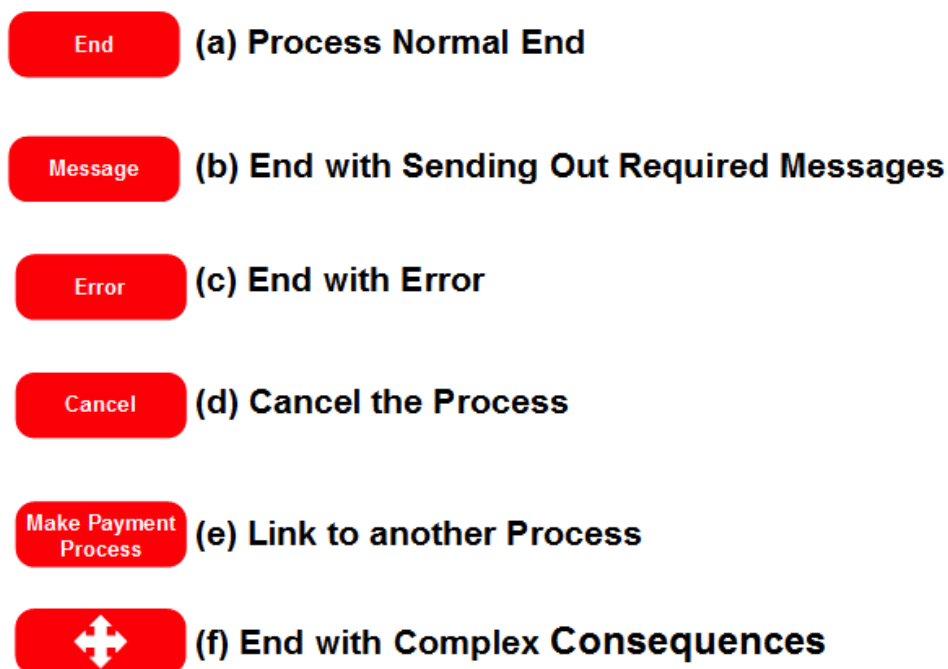
	<ul style="list-style-type: none"> <li>• Predefined condition description (Double Sided Arrow), could be: <ol style="list-style-type: none"> <li>1. <b>Message:</b> only one (or more) specified messages can start this Process</li> <li>2. <b>Time:</b> The process starts at a certain time</li> <li>3. <b>Process Id:</b> This is a sub-Process of some other process. This process will start only when its parent process is complete.</li> <li>4. <b>Others:</b> the complex start condition specification. The process will start only when a list of user predefined conditions is satisfied.</li> </ol> </li> </ul>
<b>Sub-Process</b>	If this process has sub-processes, record their process IDs. Otherwise, the value is None.
<b>Parent-Process</b>	If this process has Parent-processes, record their process IDs. Otherwise, the value is None.
<b>Data</b>	This area is used to represent the incoming data from other processes. It can be none if there is no data communication.

### 3.3.2 Process End

The Process End notation defines the result that is a consequence of a Process Flow ending. In some circumstances, multiple types of results that can be defined in Process End stage. The Process End notation indicates that a process will end and there is no outgoing Process Flow that connects from a Process End icon. A Process End notation is a rounded corner rectangle with both outline and fill colour being red. The notations also have an open centre so that the end results can be included within the shape. The description of the result is placed in the centre of rectangle boundary in white font. If there are multiple consequences belonging to the same Process End, a white quad arrow icon is placed in the centre of the shape.



Figure 3.7 shows the different aspects of the Process End notation (with different consequences). Figure 3.7 (a) is a Process End notation in a normal end condition without further information transfer. A white key word *End* appears in the centre of the red rounded-corner rectangle. If a process is required to send messages in its normal ending state, the key word *Message* will be used to replace *End* (as shown in (b)). Figure 3.7 (c) shows that the process terminates with errors and (d) means the process is cancelled by the user. If the end of one process leads to the start of another, the name of the new process will be represented in the shape. Figure 3.7 (e) shows this; the *Make Payment Process* will start immediately after the existing process finishes. If the process leads to other complex results, a quad arrow appears in the Process End icon (as shown in (f)). The complex consequences could be several user pre-defined rules, a list of complex conditions or an integration of some results from the process. The end of this process will result in all of these consequences.



**Figure 3.7: Process End Notation**

The common attributes of a Process End notation include:

Attributes	Description
<b>Id</b>	A unique Id that distinguishes the Process End notation from other notations.
<b>End Type</b>	A process end type must be Normal, Message, Error, Cancel, Link to another process or Other Complex Consequences
<b>Message</b>	If the Process End leads to a Message sending, then the Message content must be supplied. It can be None if the notation is another type.
<b>Error</b>	If the Process ends in an error on this flow, then the error information must be supplied. It can be None if the notation is another type.
<b>Cancel</b>	If the Process is cancelled, then the output information must be supplied. It can be None if the notation is another type.
<b>Sub-Process Name</b>	If the Process End leads to another Process the name of the new sub-Process must be supplied. It can be None if the notation is another type.
<b>Sub-Process ID</b>	If the Process End leads to another Process, then the ID of the new sub-Process must be supplied. It can be None if the notation is another type.
<b>Sub-Process Data</b>	If the Process End leads to another Process, then the data to be communicated to the new sub-Process must be supplied. It can be None if the notation is another type or there is no data communicated to the new process.
<b>Complex Results</b>	If the Process End leads to multiple consequences, all the relevant results will be listed here. It can be None if the notation is another type.

### **3.3.3 Data Encapsulation**

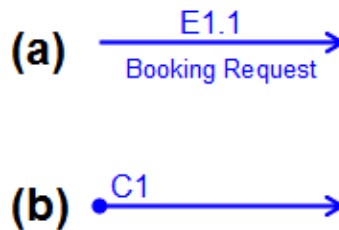
In order to enhance the readability and reduce diagram complexity, EML does not use “Data Flow”. All the data communicated between services, operations and processes are encapsulated in the flows, service nodes and operation nodes. By using this mechanism, the user can focus on the business process itself while complex data details are hidden behind it. However, the user can always obtain (and show) this information from the notation’s data attributes. It provides a more flexible and clearer view to the business processes and service structures.

### **3.3.4 Process Flow**

A Process Flow is used to show the order of operations that will be performed in a Process. Each Process Flow has only one source and one target. The source and target must be retrieved from the following set of Process Objects: Process Start, Process End, Operations, Services / sub-Services, and Conditions. A process flow edge from node A to node B is used to show that operation B cannot start until operation A has reached a certain execution state associated with the edge. Examples of such states are: finished (by default), failed (when an error is detected during the execution of the task), aborted (after a user has killed the task), skipped (when the operation has been skipped) or other (other unexpected states) etc. The state is visually represented by the color of the operation (or service / sub-service) boundary positioned at the tail of the process flow edge or the exception handler icon in the center of the target notation shape. This makes it easy to follow, at runtime, whether a process flow dependency has been activated.

A Process Flow is a line with a solid arrowhead drawn in blue color. Each flow has a unique process sequence ID and the user can show or hide this at any time. The encapsulated data for the process flow is hidden by default. But the user can change it to visible if required. If the process flow is a default path (normally connects to a

condition), a dot icon will appear at the start of the flow. Figure 3.8 (a) represents a typical Process Flow with its Process Sequence ID (*E1.1*) on top of the flow and the communication data (*Booking Request*) under the flow. However, the default setting for the sequence ID and Data are all invisible. Figure 3.8 (b) is a default process flow where the sequence ID and data are hidden. A blue dot marker is shown at the beginning of the flow line. The start condition ID of this flow (*C1*) appears at the beginning of the process flow.



**Figure 3.8 (a): A Process Flow with Sequence ID and Data; (b): A Default Process Flow**

The common attributes of a Process Flow include:

<b>Attributes</b>	<b>Description</b>
<b>Id</b>	A unique Id which distinguishes the Process Flow notation from other notations.
<b>Name</b>	A textual name for the Flow.
<b>From</b>	Identifies the incoming object of this Process Flow
<b>To</b>	Identifies the outgoing object of this Process Flow
<b>Flow Start Condition ID</b>	If the flow has a start condition, record its unique condition ID here
<b>Flow Start Conditions</b>	The start conditions of the flow. This can be None if there are no start conditions.
<b>Sequence ID</b>	A unique ID that distinguishes the process sequence for this flow.
<b>Data</b>	An optional attribute that identifies the encapsulated data package that is being sent. It can be None if there is no data

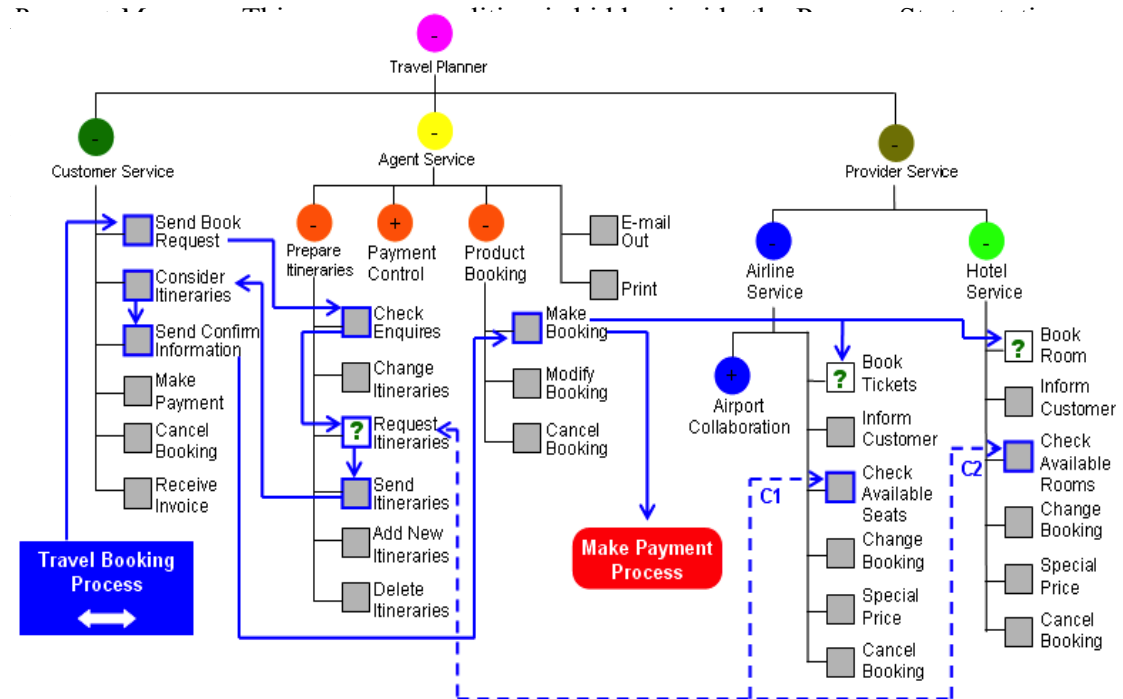
	communication.
<b>Default Status</b>	If it is a default flow, set the value to True. Otherwise, the value is False
<b>Iteration</b>	If the process flow is involved in a loop, set the value to be true. Otherwise, the value is false
<b>Loop Times</b>	The loop times in the process. The value is an integer (from 0 to any positive number).
<b>Loop Start Conditions</b>	The start conditions of the loop.
<b>Loop Start Condition ID</b>	The unique loop start condition ID.
<b>Loop End Conditions</b>	The end conditions of the loop.
<b>Loop End Condition ID</b>	The unique loop end condition ID.

### 3.3.5 Business Process Layer

Figure 3.9 shows a *Travel Booking Process* in an EML process overlay. Only process related services and operations are shown; other, unrelated services have been elided (e.g. *Payment Control Service*, *Airport Collaboration Service*). The process starts (blue rectangle) with a client side application passing a request message to the *Send Book Request* operation of the *Customer Service*. The *Agent Service* receives the request through the *Check Enquires* function, and uses its *Request Itineraries* operation to check availability information with the *Airline* and *Hotel services*. The agent requests flights and rooms with a list of parameters. There are iterations (dashed double arrowheads links) between *Request Itineraries*, *Check Available Seats* and *Check Available Rooms* (please refer to Section 3.5 for detailed information). When the agent finds that both the air ticket and the hotel room are available on the

requested date (end condition C1 & C2), it terminates the loop and sends the client a report generated by the Send Itineraries operation. The customer *Considers Itineraries* and *Sends Confirm Information* to the *Agent Service*. The agent receives this information and then *Makes Booking*. After both of the *Book Tickets* and *Book Room* operations are successfully completed, the agent calls *Make Payment Process* (Figure 3.14) to ask for the payment and end the existing process (Rounded Rectangle).

The double-sided arrow in the Process Start indicates that the process needs to start with a condition. In this example, the Process Start condition is the arrival of *Booking*



**Figure 3.9: Travel Booking Process Overlay**

There are two types of execution states included in this example:

- (1) The finish status (by default). In this state, the boundaries of the operations have been changed to a blue colour. It means that this process step is completed successfully.
- (2) The failed status. In this state, a green question mark appears in the centre of the operation shape (e.g. in *Request Itineraries, Book Tickets and Book Room* operations). It means that the process step with this operation is not complete normally.

The common attributes of a Business Process Layer include:

<b>Attributes</b>	<b>Description</b>
<b>Id</b>	A unique Id that distinguishes this Process from processes.
<b>Name</b>	A textual name for the process.
<b>Version</b>	The Version number of this Process Diagram.
<b>Author</b>	The author of this Process Diagram.
<b>Language</b>	The language in which text is written. The default is English.
<b>Service Trees</b>	An EML Process may contain one or more service trees. The IDs of the Service trees are recorded here.
<b>Documentation</b>	The process may have optional text documentation to describe it.
<b>Process Type</b>	The type must be Process or sub-Process
<b>Process State</b>	The Status of a Process is determined when the Process is being executed. The full list of states include: None, Ready, Active, Finish, Failed, Aborting, Aborted, Skipping, Skipped, Cancelling, Cancelled and Other
<b>Process Flows</b>	Identifies all of the Process Flows (IDs) that are contained within the Process.
<b>Operations</b>	Identifies all of the Operations (IDs) that are contained within the Process.
<b>Dependency</b>	Identifies all of the Dependency Triggers (IDs) that are

	contained within the Process.
<b>Exceptions</b>	Identifies all of the Exception Handlers (IDs) that are contained within the Process.
<b>Sequences</b>	The sequence of all the notations, flows, operations, dependencies and exceptions.

### 3.4 Exception Handler Overlay

In EML, an exception handler is another overlay mechanism designed to handle the occurrence of some condition that changes the normal process flow of execution. The condition is called an exception. Modelling failure-handling behaviour is an important requirement for a visual modelling language, as exceptions are common when running processes in a distributed environment. Raising an exception is a useful way to signal that the process or operation could not execute normally, for example when its input parameters are invalid (e.g. empty message or wrong date format) or when a resource it relies on is unavailable (no spare rooms available anymore, or the customer cannot pass the credit check). In systems without exceptions, the process would need to return some special error code and abort itself. However, this simple solution is sometimes not adequate to tackle the complex business process problem. The users often require a comprehensive and flexible method to cope with exceptions.

A common way to solve the exception problem in most existing process modelling languages is based on a rollback method. The current state is saved in a predefined location and execution switches to a predefined handler. Depending on the situation, the handler may later resume execution at the original location, using the saved information to restore the original state. For example, BPMN has an important notion: business transactions. A transaction here means a collection of activities that must be performed "atomically" – all must complete successfully – or else the system must be "rolled back" to its initial state, as if none of them had ever occurred.



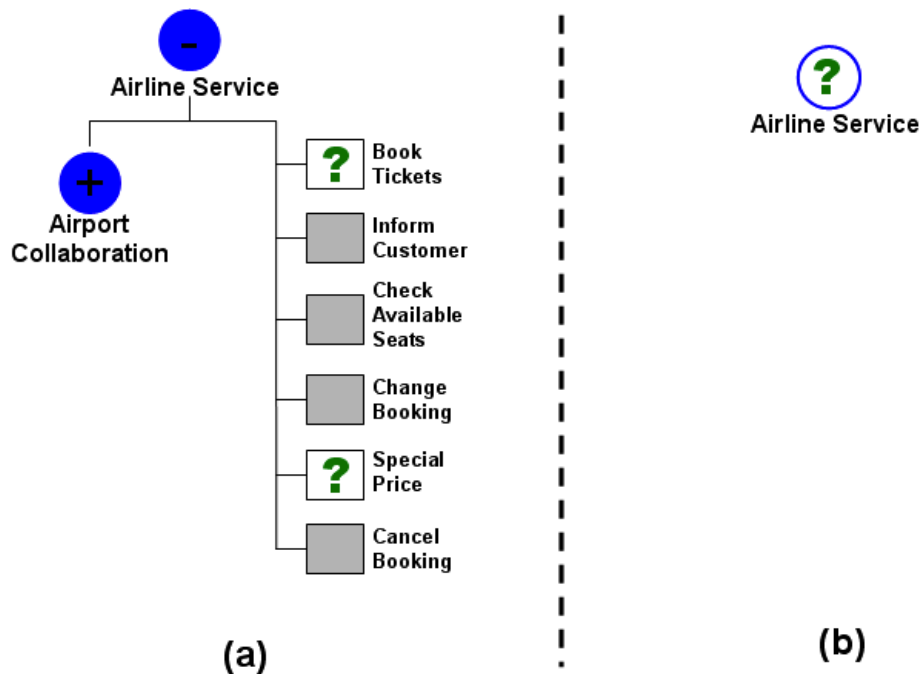
However, one of the major problems for this approach is its scalability. In this exception handling method, all the resources performing each part of the transaction must be "locked" until the transaction either commits or rolls back. As the process size becomes bigger, business transactions become too long to be able to lock the resources, making it very hard to roll back to the initial state.

On the other hand, for a complicated process, a single approach is usually not sufficient to model exception handling. Using travel booking as an example, consider that you are booking a trip to Europe online. Neither the airline nor the hotel can guarantee either price or availability until you actually book specific dates with a credit card. After we successfully reserved hotel rooms, we found that the air tickets on the required date are no longer available. The traditional business transaction will cancel all the air ticket and hotel booking and roll back to the start stage and rebook all the itineraries again. Obviously, this is not the best solution. The exception handler should have the ability to negotiate alternative solutions with the users. For example, if the economy class tickets are sold out, in addition to normal transactions, the exception handler may ask the users whether they want to purchase an available business class ticket or use their air points to upgrade. For this reason, we designed the exception handler overlay in EML. Instead of rolling back, an exception handling layer is constructed to model transaction error handling in much more details.

### **3.4.1 Failure Handling Notation**

A failure handler annotation is a green question mark in the middle of an operation or service. Despite the original state of an operation or service, as soon as an exception handler occurs in it, the related operation or service's fill colour must be changed to white and the boundary set to a single line. If the user removes the handling notation, the operation or service changes to its original state automatically. This annotation is used to specify a transaction failure. Users can establish several start conditions in the properties to discriminate between different kinds of failures and activate appropriate

exception handlers. Figure 3.10 shows two different usage situations of the failure handler annotation. Figure 3.10 (a) shows the exception handler in *Book Tickets* and *Special Price* operations. It represents there are one or more exception handling solutions relating to the operations. The users can check the detailed information about these handlers in the exception layer. Figure 3.10 (b) shows the failure handling notation in the elided *Airline Service* node. If a failure handling notation appears in a service node, it means that at least one operation or sub-service in this service has an exception handler.



**Figure 3.10 (a): Failure Handling Notations in Operations;**  
**(b): Failure Handling Notation in Service**

In the same service tree, the exception handling notation will only appear either in a service node or in its operations. For example, because the *Book Tickets* operation is inside *Airline Service*, in Figure 3.10 (a) when the *Airline Service* tree is expanded, the user will see the failure handling notations in *Book Tickets* and *Special Price* operations (and there is no green question mark in *Airline Service* node). However, if we collapse the *Airline Service* tree (as in Figure 3.10 (b)), all the operations and

sub-services inside this *Airline Service* will be hidden. In this case, the users will only find a green question mark in the service node to represent all the exception handlers in its operations and sub-services. In this case the elision symbol (+) is replaced by failure handling notation (green question mark).

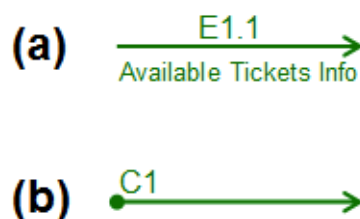
The common attributes of a Failure Handler annotation include:

<b>Attributes</b>	<b>Description</b>
<b>Id</b>	A unique Id that distinguishes the Failure Handling notation from other notations.
<b>Name</b>	A textual name for the exception handler.
<b>Start Condition List</b>	The start conditions that discriminate different kinds of failures and activate appropriate exception handlers. An operation failure handler annotation needs to have at least one start condition. The value can be None if the Failure Handler is in a service.
<b>Exception Handler List</b>	Matchning the start conditions, a list of corresponding exception handlers must be defined here. An operation failure handling notation need to have at least one exception handler. The value can be None if the Failure Handler is in a service.
<b>Sub Exceptions</b>	Records the exception handler IDs of a Service's operations and sub-services. The value can be None if the Failure Handler is in an operation.
<b>Data</b>	Used to specify the incoming data from other processes. It can be none if there is no data communication.

### 3.4.2 Exception Flow

An exception flow is used to show the sequencing between operations that the exception handler will perform. Each exception flow has only one source and one

target. The source and target must be retrieved from the available set of the Operations or Services / sub-Services. An Exception Flow is a green line with a solid arrowhead. Each flow has a unique exception handler sequence ID which the user can show or hide any time. The encapsulated data for the exception flow is hidden by default. But the user can change it to visible if required. Five different exception handling execution states are visually represented by the colour of the Operation and Service / sub-Service boundary positioned at the tail of the exception flow edge or the exception handler and status icon in the center of the target notation shape. Figure 3.11 (a) represents a typical Exception Flow with its Exception Sequence ID (*E1.1*) on top of the flow and the communication data (*Available Ticket Info*) under the flow. However, the default setting for the sequence ID and Data are all invisible. If an exception flow is a default flow, a green dot mark appears at the beginning of the line (as shown in Figure 3.11 (b)). The start condition of the flow (*CI*) appears at the beginning of the exception flow.



**Figure 3.11 (a): An Exception Flow with sequence ID and Data;**  
**(b): A Default Exception Flow**

The common attributes of a Process Flow include:

Attributes	Description
<b>Id</b>	A unique Id that distinguishes the Exception Flow notation from other notations.
<b>Name</b>	A textual name for the Flow.
<b>From</b>	Identifies the incoming object of this Exception Flow.
<b>To</b>	Identifies the outgoing object of this Exception Flow.
<b>Flow Start</b>	If the flow has a start condition, record its unique condition ID

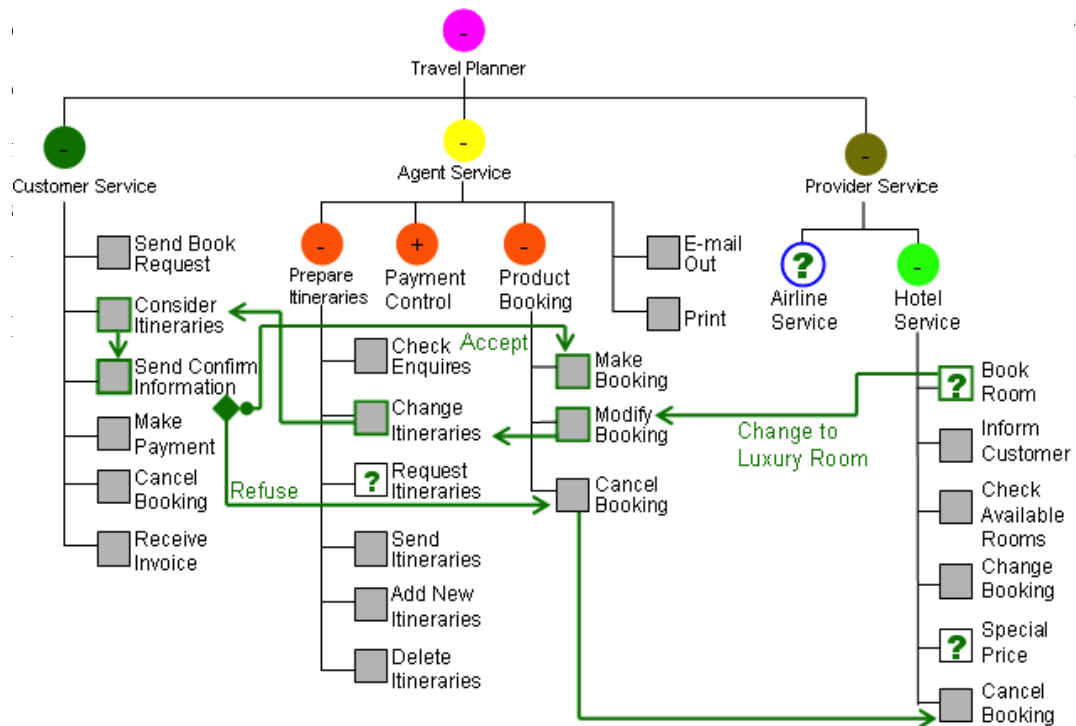
<b>Condition ID</b>	here.
<b>Flow Start Conditions</b>	The start conditions of the flow which can be None if there are no start conditions.
<b>Sequence ID</b>	A unique ID that distinguishes the exception handler sequence for this flow.
<b>Data</b>	An optional attribute that identifies the encapsulated data package that is being sent. It can be None if there is no data communication.
<b>Default Status</b>	If it is a default flow, set the value to True. Otherwise, the value is False.
<b>Iteration</b>	If the exception flow is involved in a loop, set the value to be true. Otherwise, the value is false.
<b>Loop Times</b>	The loop times in the exception handler. The value is an integer (from 0 to any positive number).
<b>Loop Start Conditions</b>	The start conditions of the loop.
<b>Loop Start Condition ID</b>	The unique loop start condition ID.
<b>Loop End Conditions</b>	The end conditions of the loop.
<b>Loop End Condition ID</b>	The unique loop end condition ID.

### 3.4.3 Exception Layer

Figure 3.12 shows a hotel room booking exception handler layer. Instead of simply calling *Cancel Booking* and rolling back, we use EML exception overlay to model a more complete exception solution. If the *Hotel* finds that all standard rooms on the required date have been booked out, it sends a negotiation message (*Change to*

*Luxury Room*) back to the travel agent. The *Agent Service Modifies Booking, Changes Itineraries* (previous travel plan), makes a new itinerary and sends to the *Customer Service*. The customer receives the latest updated travel plan and *Considers Itineraries*. When they make the final decision, the *Customer Service* then *sends Confirm Information* to the *Agent Service* again. If the user *Accepts* the hotel's suggestion, the process will lead to *Make Booking* again. Otherwise (*Refuse*), the customer informs the agent to *Cancel Booking*. The *Agent Service* asks the *Hotel Service* to *Cancel Booking*.

The green diamond icon after *Send Confirm Information* is used to represent the



**Figure 3.12: Hotel Room Booking Exception Handler Overlay**

There are another three exception handling icons in *Special Price* and *Request Itineraries* operations and *Airline Service*. The user defines the detailed exception handler processes in different layers, but they can combine them with this example handler in the same layer if desired. Even for the same exception handling notation, the user can define several different exception layers by their start conditions. So in EML, the exception handling is much more than a simple roll back mechanism. It is an individual process or even a complicated integration of alternative processes. The exception handling overlay is an individual overlay based on the same service tree structure as conventional processes. EML has the freedom to allow the user to define them in a single layer (as shown in Figure 3.12) or combine them with the process and trigger overlays to generate an integrated overview of the system (as to be shown in Chapter 5).

The common attributes of an Exception Layer include:

<b>Attributes</b>	<b>Description</b>
<b>Id</b>	A unique Id that distinguishes this exception layer from other overlays.
<b>Name</b>	A textual name for this exception layer.
<b>Version</b>	The Version number of this exception handler.
<b>Author</b>	The author of this exception handler.
<b>Language</b>	The language in which text is written. The default is English.
<b>Service Trees</b>	An EML exception handling layer may contain one or more service trees. The Service trees' ID's are recorded here.
<b>Documentation</b>	The process may have optional text documentation to describe it.
<b>Exception Handler Status</b>	The Status of an exception handler is determined when the handler is being executed. The full list of states include: None, Ready, Active, Finish, Failed, Aborting, Aborted, Skipping, Skipped, Cancelling, Cancelled and Others.
<b>Start Conditions</b>	The start conditions of this exception handler.

<b>Exception Flows</b>	Identifies all of the Exception Flows (IDs) that are contained within the overlay.
<b>Operations</b>	Identifies all of the Operations (IDs) that are contained within the exception overlay.
<b>Dependency</b>	Identifies all of the Dependency Triggers (IDs) that are contained within the exception overlay.
<b>Exceptions</b>	Identifies all of the other Exception Handlers (IDs) that are contained within this exception overlay.
<b>Sequences</b>	The sequence of all the notations, flows, operations, dependencies and exceptions.
<b>Parent Processes</b>	An exception handler may be used by several different processes. The parents IDs are recorded here.

### 3.5 Dependency / Trigger

It is important to know if a specific event occurs or a condition is met. Events and conditions are referred to as dependency relationships. In some cases, we can also treat internal (system) exceptions as triggers. BPMN does not distinguish the internal and external dependency. They are all represented as flows in the diagram. It increases the complexity of the diagram. An EML trigger layer can be used to solve dependency problems. A trigger overlay is another layer in EML which is specially used to model system internal dependencies. Since EML uses a multi-layer structure, users can choose to combine the trigger layer with the process layer (as in Figure 3.14) or separate them by using different views to reduce diagram complexity.

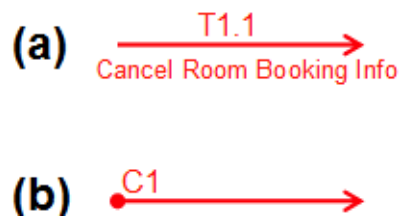
In general, a trigger is a special case of a process. The major difference between a process and a trigger is the actor. A process is performed by a user; the sequence and result of a process are normally variable. A trigger is enacted by the system itself (automatically). Because it is an internal system dependency, the trigger execution



order and outcomes (for the same trigger condition) are usually unalterable. Thus another benefit to use the trigger overlay in EML is to support internal dependency process reuse. As long as a trigger is defined, it can be automatically reused in all different processes and exception handlers.

### 3.5.1 Trigger Flow

A trigger flow is used to show the sequence that a system trigger (or internal dependency) will be performed. Each exception flow has only one source and only one target. The source and target must be from the set of the Operations or Services / sub-Services. Similar to process and exception flows, an Exception Flow is a line with a solid arrowhead, but colored red. Each flow has a unique trigger sequence ID and the user can show or hide this at any time. The encapsulated data for the trigger flow is hidden by default, but the user can change it to visible if required. Again, five different system trigger execution status are visually represented by the color of the operation and service / sub-Service boundary positioned at the tail of the trigger flow edge or the exception handler and status icon in the center of the target notation shape.



**Figure 3.13 (a): A Trigger Flow with sequence ID and Data; (b): A Default Trigger Flow**

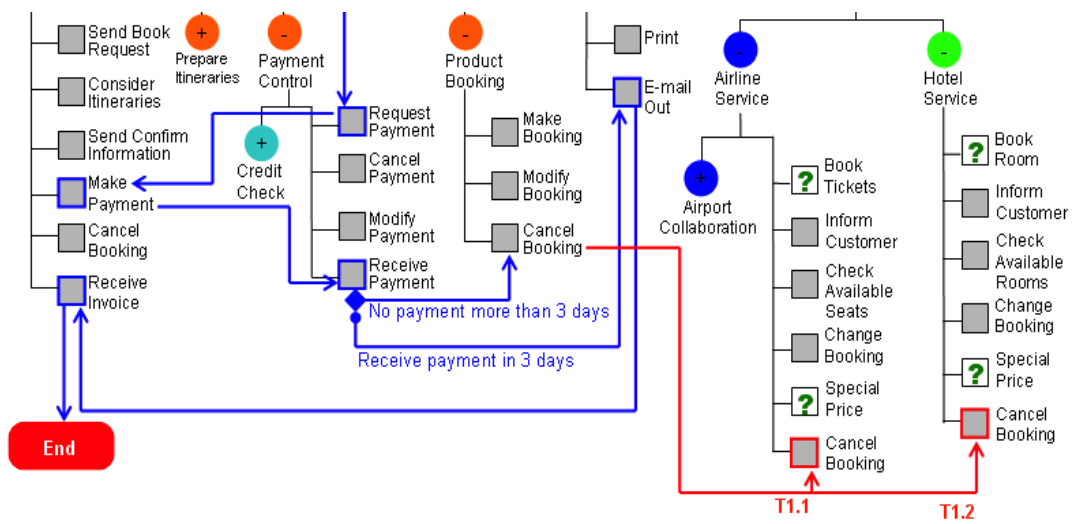
Figure 3.13 (a) shows a typical Trigger Flow with its Trigger Sequence ID (*T1.1*) on top of the flow and the communication data (*Cancel Room Booking Info*) under the flow. However, the default setting for the sequence ID and Data are all invisible. If a trigger flow is a default flow, a red dot mark appears at the beginning of the line (as shown in Figure 3.13 (b)). The start condition of the flow (*C1*) appears at the beginning of this trigger flow.

The common attributes of a Trigger Flow include:

<b>Attributes</b>	<b>Description</b>
<b>Id</b>	A unique Id that distinguishes the Trigger Flow notation from other notations.
<b>Name</b>	A textual name for the Flow.
<b>From</b>	Identifies the incoming object of this Trigger Flow.
<b>To</b>	Identifies the outgoing object of this Trigger Flow.
<b>Flow Start Condition ID</b>	If the flow has a start condition, record its unique condition ID here.
<b>Flow Start Conditions</b>	The start conditions of the flow. It can be None if no start conditions.
<b>Sequence ID</b>	A unique ID that distinguishes the trigger sequence for this flow.
<b>Data</b>	An optional attribute that identifies the encapsulated data package that is being sent. It can be None if there is no data communication.
<b>Default Status</b>	If it is a default flow, set the value to True. Otherwise, the value is False.
<b>Iteration</b>	If the trigger process flow is involved in a loop, set the value to be true. Otherwise, the value is false.
<b>Loop Times</b>	The loop times in the trigger. The value is an integer (from 0 to any positive number).
<b>Loop Start Conditions</b>	The start conditions of the loop.
<b>Loop Start Condition ID</b>	The unique loop start condition ID.
<b>Loop End Conditions</b>	The end conditions of the loop.
<b>Loop End Condition ID</b>	The unique loop end condition ID.

### 3.5.2 Trigger Overlay

Figure 3.14 illustrates the *Make Payment Process* example with trigger flows. It follows the *Travel Booking Process* example from Figure 3.9. When the user successfully books air tickets and hotel rooms from *Travel Booking Process*, the *Agent Service* starts to *Request Payment* using *Payment Control* service. The agent sends the payment request to *Customer Service*, and the customer then *Makes Payment*. If the agent *Receives Payment* within three days (default option), then it sends the invoice by *E-mail* to the customer. The customer *Receives the Invoice* and the whole process ends. However, if the agent *Payment Control* service doesn't receive the payment in three days, it *Cancel the Booking* using *Product Booking* service. This operation triggers two extra operations automatically: *Cancel Booking* in *Airline Service* (T1.1) and *Hotel Service* (T1.2). In this example, we integrate the trigger with the process overlay. However, the user can hide the trigger and define it in a different layer to reduce the diagram complexity. The *Cancel Booking* trigger itself is reusable. So for any other processes, an exception handler or trigger can be directly reused by linking the flow to the operation and filling in a trigger start condition.



**Figure 3.14: Make Payment Process with Triggers**

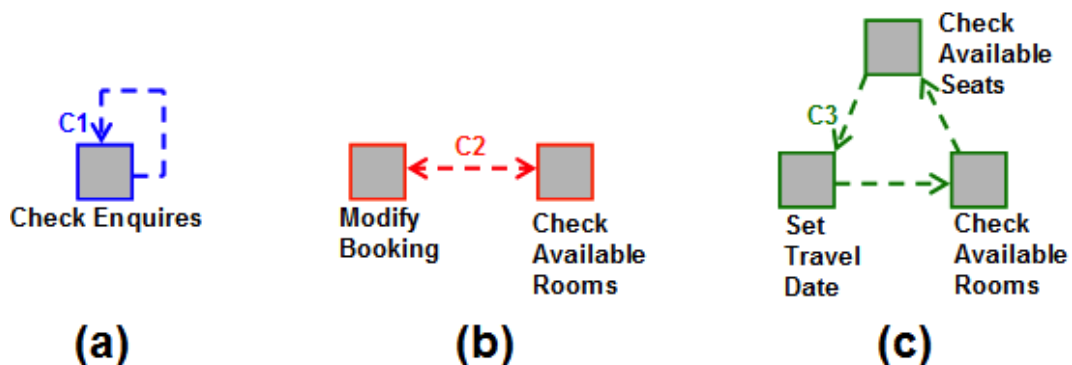
The common attributes of a Trigger Layer include:

Attributes	Description
<b>Id</b>	A unique Id that distinguishes this trigger from other processes.
<b>Name</b>	A textual name for this trigger.
<b>Version</b>	The Version number of this trigger.
<b>Author</b>	The author of this trigger.
<b>Language</b>	The language in which text is written. The default is English.
<b>Service Trees</b>	An EML trigger layer may contain one or more service trees. The Service trees' IDs are recorded here.
<b>Documentation</b>	The process may have optional text documentation to describe it.
<b>Start Conditions</b>	The start conditions of this trigger.
<b>Trigger Flows</b>	The Trigger Flows (IDs) that are contained within the overlay.
<b>Operations</b>	The Operations (IDs) that are contained within the trigger overlay.
<b>Dependency</b>	The Dependency Triggers (IDs) that are contained within the trigger overlay.

<b>Exceptions</b>	The other Exception Handlers (IDs) that are contained within this trigger overlay.
<b>Sequences</b>	The sequence of all the notations, flows, operations, dependencies and exceptions.
<b>Invoker IDs</b>	A trigger may be used by several different processes, exception handlers or other triggers. The invokers' IDs are recorded here.

### 3.6 Iteration

EML supports specification of process iteration at different levels. The iteration flow is visually represented by broken lines. BPMN uses a loop icon in a task to represent iteration. However, if the user wants to represent the iteration between multiple tasks, he has to use extra components (e.g. pool and group etc.) to model it. In EML, we represent iterations occurring in different overlays by using the same visual method (styled lines), and use different process specific colours to distinguish them. This increases the consistency of the notation and reduces the modelling complexity of the diagram. For example, the iteration in a process overlay is modelled by changing the process flow from a solid line to a dashed line (as shown in Figure 3.15 (a)). For an exception handler, a solid exception flow is changed to a broken line flow (as shown in Figure 3.15 (c)), and likewise for trigger flows (as shown in Figure 3.15 (b)).



**Figure 3.15 (a): Loop in Process Overlay with Single Activity;**  
**(b): Loop in Trigger Overlay with Two Operations;**

### (c): Loop in Exception Handler Overlay with Three Operations

There are three types of loops:

1. A single activity loop is represented as a single arrowhead dashed line whose source and target are same operation (or service/sub-service). Attributes in the dashed flow control the iteration (e.g. loop times, start and complete conditions, input/output data etc.). *Check Enquires* in Figure 3.15 (a) is a single activity loop example in process overlay. The *Prepare Itineraries* service inside travel *Agent Service* uses this iteration to check all the travel related enquiries from the *Customer Service* (e.g. all the available travel dates, special air tickets, possible hotel room promotions etc.). The loop keeps working until an end condition *CI* (all the enquiries have been answered) is satisfied.
2. Loops of two operations (or services / sub-services), use a dashed line with two arrowheads. Figure 3.15 (b) shows the iteration of the *Modify Booking* and *Check Available Rooms* operations in the trigger overlay. When the *Agent Service* received the room booking application from the *Customer Service*, they need to check whether the suitable hotel room is available on customer required date. The customer normally sends a list of preferred date and room type for the room booking. The *Agent Service* starts from the highest preference date and room type to the lowest preference date and room type. The process loops until a termination condition *C2* is met (finds the suitable room on required date or there is no suitable room available on all the preferred dates).
3. If a loop involves more than two operations (or services / sub-services), a single arrowhead dashed line guides direction, linking different operations or services in a closed circuit. Figure 3.15 (c) represents an iteration example among three different operations in exception handler overlay. The agent *Sets*

*Travel Date* based on customer's requirements, then *Check Available Rooms* from the hotel and *Check Available Seats* from the airline. If it cannot find room and air ticket on same date, the agent then *Reset Travel Date* and start search again. The loop keeps working among these three operations until there are room and air tickets both available on the required date, or if it cannot find them on one of the other customer preferred dates (C3).

The attributes of the iterations are combined in all the flow objects. Please check process flow, exception flows and trigger flows' attributes for more detailed iteration related information.

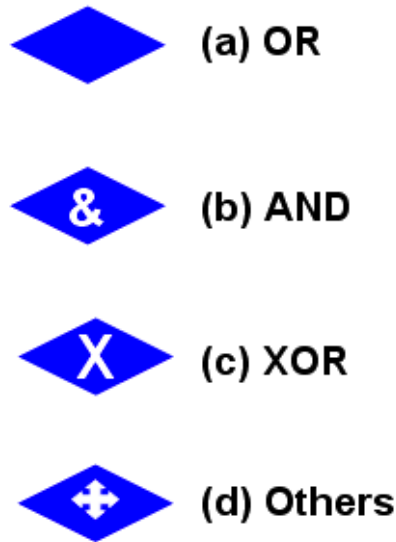
### **3.7 Conditions**

Conditions are modelling elements that are used to control how flows interact as they converge and diverge within a process, exception handler or trigger. The term "Conditions" implies that there is a gating mechanism that either allows or disallows passage through the gate. The traditional diamond shape has been widely accepted as a visual representation for condition (in flow chart diagram). Hence we decide to adopt similar approach in our EML. When the flows arrive at a condition, they can be merged together on input and/or split apart on output as the condition mechanisms are invoked. To be more descriptive, a condition is actually a collection of "Logical Gates." There are different types of conditions (as described below) and the behaviour of each type of conditions will determine the approach for the continuation of incoming flows.

In EML, a condition shape is a diamond. The fill colour of the condition is based on the overlays. If a shape is used in a process layer, it appears in blue. However, if it is

in an exception handler or trigger overlay, then the colour becomes green or red. All conditions have an open centre so that the condition type icons can be included within the shape to help distinguish the condition groups (e.g. OR, AND, XOR or OTHERS).

Figure 3.16 lists all the condition types in EML process overlay. Figure 3.16 (a) is the default setting of a condition. It is used to represent OR relationships between the incoming flows. The AND relationship is represented by a “&” sign in the middle of the diamond shape (as shown in Figure 3.16 (b)). In Figure 3.16 (c), a “X” icon appears in the condition shape, it means the XOR relationship. If the user defines any extra complex conditions, a white quad arrow is used to represent them.



**Figure 3.16: Conditions in Process Overlay**

The common attributes of a Condition include:

Attributes	Description
<b>Id</b>	This is a unique Id that distinguishes this condition from other notations.
<b>Overlay</b>	A condition can be used in Process, Exception Handler or Trigger Overlays.
<b>Condition Type</b>	A condition type can be OR, AND, XOR or Others.



<b>Language</b>	This holds the language in which text is written. The default is English.
<b>Incoming Flows</b>	All the incoming flow IDs.
<b>Incoming Conditions</b>	If there are Multiple incoming Flows, Incoming Conditions expression must be set in here. This will consist of an expression that can reference incoming Flows and (or related Data).
<b>Outgoing Flows</b>	All the outgoing flow IDs.
<b>Start Conditions</b>	The start conditions of this trigger.
<b>Outgoing Conditions</b>	If there are Multiple outgoing Flows, Outgoing Conditions expression must be set here. This will consist of an expression that can reference (outgoing) Flows and (or Data).
<b>Pre-Defined Conditions</b>	If the condition type is OTHERS, list all the user defined conditions in here.

### 3.8 Summary

In this chapter we have described EML, a novel business process modelling language based on tree hierarchies and tree overlay metaphors. The service architectures are represented as trees and the business sequences are modelled as process overlays on the service trees. By combining the above two mechanisms EML gives users a clear overview of a whole enterprise system while all the business processes are modelled by overlays on the same view. It successfully weaves service-oriented and process-oriented methods into the same visual language.

EML uses a multilayer structure to model business processes, exception handlers and dependency triggers in different levels. This approach potentially reduces the complexity of business processes. Our objective with EML is to develop an easy to understand visual specification mechanism for both business and technical users. It will mitigate the limitations of existing visual modelling notations and bridge the gap

between business design, implementation and integration. The long term contribution for this research is to apply our novel EML notation to maximize the simplicity and efficiency of enterprise process integration, and to automatically map to and from business process execution environments. Our approach does not exclude existing modelling notations. We aim to incorporate them into our EML support tool while providing additional richer, integrative views for enterprise process modelling.

## Chapter 4

# EML MODELLING TOOL IMPLEMENTATION

---

We have developed an Eclipse-based integrated design environment, MaramaEML for creating EML specifications. This IDE provides a platform for efficient visual EML model creation, inspection, editing, storage, model driven code generation, and integration with other diagram types. A distortion-based fisheye and zooming function has also been implemented to enhance MaramaEML's navigability for complex diagrams. MaramaEML also facilitates BPEL code to be automatically generated from graphical EML representations and map it to LTSAS for validation.

In this chapter we describe the implementation of our EML modelling tool prototype. The prototype was initially implemented using the standalone Pounamu metatool (Zhu and Grundy et al, 2007), and then migrated to the Eclipse-based Marama (Grundy and Hosking et al, 2006) framework, and finally redeveloped using the Marama meta-tools (Li and Hosking et al 2008), which support easy tool structure specification using a visual approach, and tool behaviour implementation via Marama's APIs and the Java programming language.

### 4.1 Introduction

The EML tool was originally created using the Pounamu metatool (Zhu and Grundy et al, 2007), which is a standalone meta-modelling environment. Various case studies and evaluations have been conducted on Pounamu to prove its meta-modelling concept and to unveil its limitations (Grundy et, al 2008; Zhu and Grundy and

Hosking et al 2006). These research results bootstrapped the development of the Marama framework (Grundy and Hosking et al 2006), which was created using the Eclipse framework exploiting its EMF and GEF plug-ins (Eclipse 2009). Marama addressed various identified limitations of Pounamu. At an early stage of Marama development, we could use Marama as an Eclipse-based editor to facilitate graphical rendering of end user models based on our Pounamu-specified tools. As Marama evolved, a set of meta-tools, including a metamodel definer, a shape designer and a view type composer, were developed to allow Marama to be independent of Pounamu. The EML tool has been rebuilt and upgraded with enhanced features using this meta-toolset.

Below we describe each of these tool iterations. In the first iteration using Pounamu, we generated a user-friendly modelling environment for EML. In the second iteration using Marama, we created MaramaEML as an improved EML modelling environment leveraging Eclipse and its plug-ins for better model and diagram management and user experience.. In the subsequent iteration using the Marama meta-tools, we focused on re-specification and re-generation of MaramaEML to incorporate BPEL generation and LTSA-based verification (Uchitel and Robert et al 2003) of EML-modelled business processes.

## **4.2 Pounamu EML Tool**

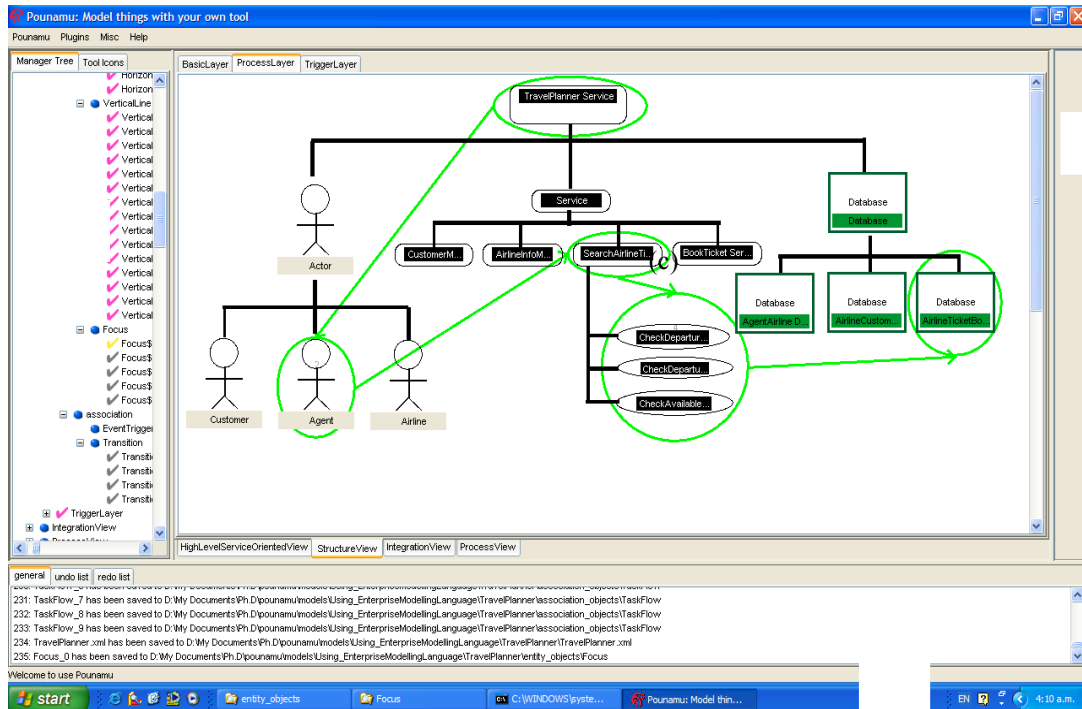
Pounamu (Zhu and Grundy et al 2007) is a standalone meta-modelling tool that allows quick construction of end user visual modelling environments. The initial exploratory EML tool icons, metamodel and views were all visually specified using Pounamu, with behaviour extensions implemented in Pounamu as Java snippets. Pounamu allowed fast development of the EML tool to prove its basic modelling concepts.

Figure 4.1 shows an exemplar EML model created in the Pounamu generated EML environment using the tree structure and process overlays. The basic tree diagram is

(a)

(b)

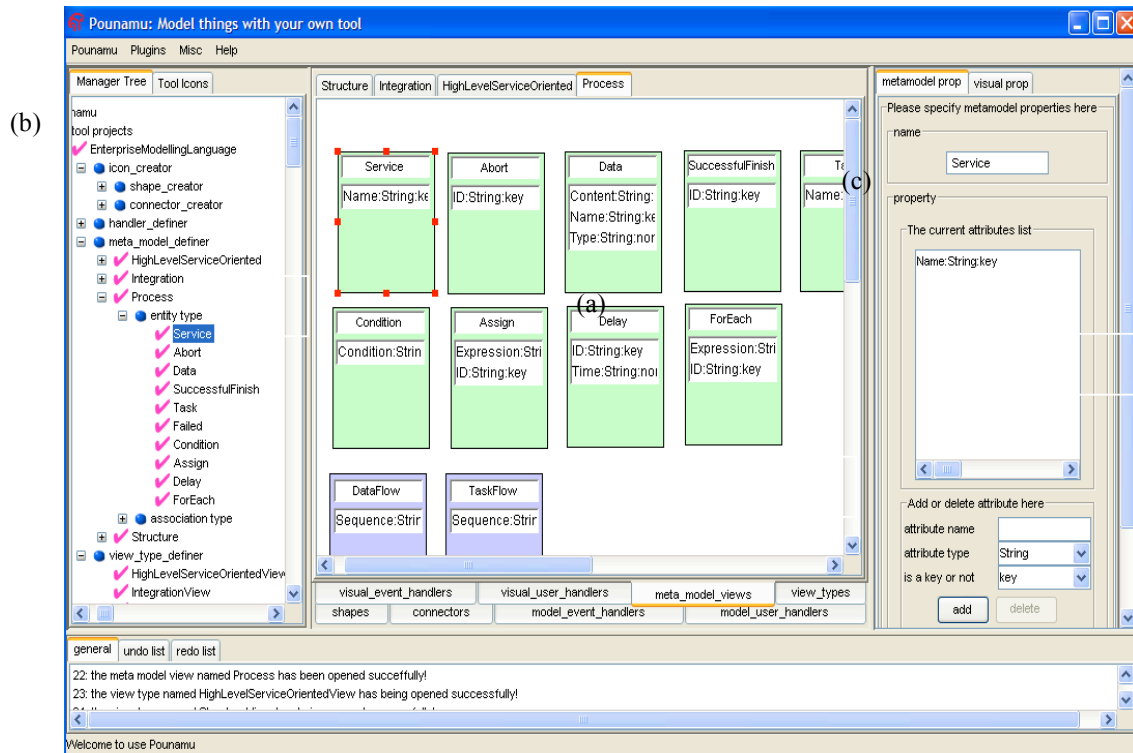
represented in (a) and all the tree nodes of EML diagram elements are organized in (b). The user can either select an element from the tree diagram (a) or from the Pounamu tree editor (b) to process future property editing. Progress outputs are shown in (c).



**Figure 4.1: The initial exploration of EML using Pounamu**

The construction of the EML tool in Pounamu comprised the following four iterative steps.

Firstly, we defined a meta-model containing entity types and relationship types, with optional key/non-key attributes. As shown in Figure 4.2, entity and relationship types were defined in a metamodel view, with properties set via a property sheet. The meta-model entities and associations are defined in Pounamu's metamodel editor (a), and they are also organized in a tree structure in the manager tree window (b). When the user selects an entity or association from (a) or (b), they can modify the properties in the property window (c).



**Figure 4.2: Construction of the EML metamodel in Pounamu**

We then defined a set of icons - shapes and connectors, to visually represent the meta-model elements. As shown in Figure 4.3 and 4.4, each shape (Figure 4.3) or connector (Figure 4.4) was defined in its own view. Complex visual properties were set via a property sheet and the changes were reflected on the icon immediately. For shape definitions (Figure 4.3), the user can define the basic shape properties (in (b)) for each component (e.g. oval, rectangle, polygon etc.), and the result will be shown in the main window (a). For connector definitions (Figure 4.4), the user defines the properties (line colour, text colour, connector start and end shapes etc.) for the connectors in (b), the reactive editing result will be directly shown in (a).

(b)

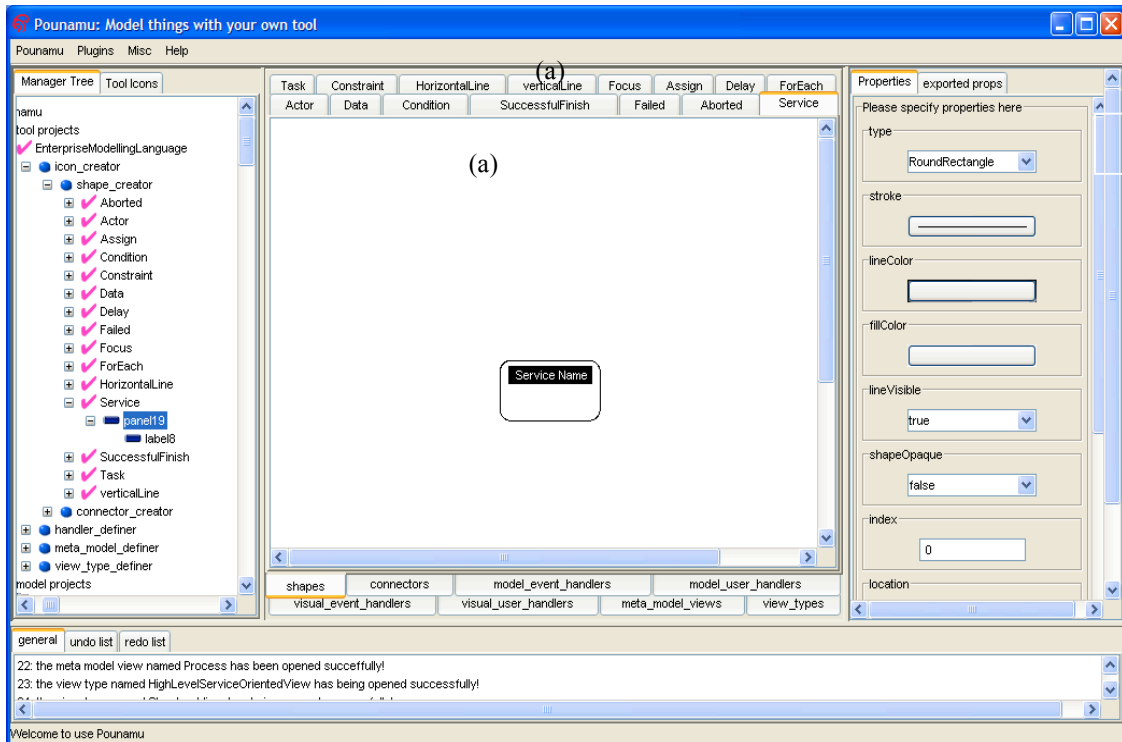


Figure 4.3: Construction of the EML shapes in Pounamu

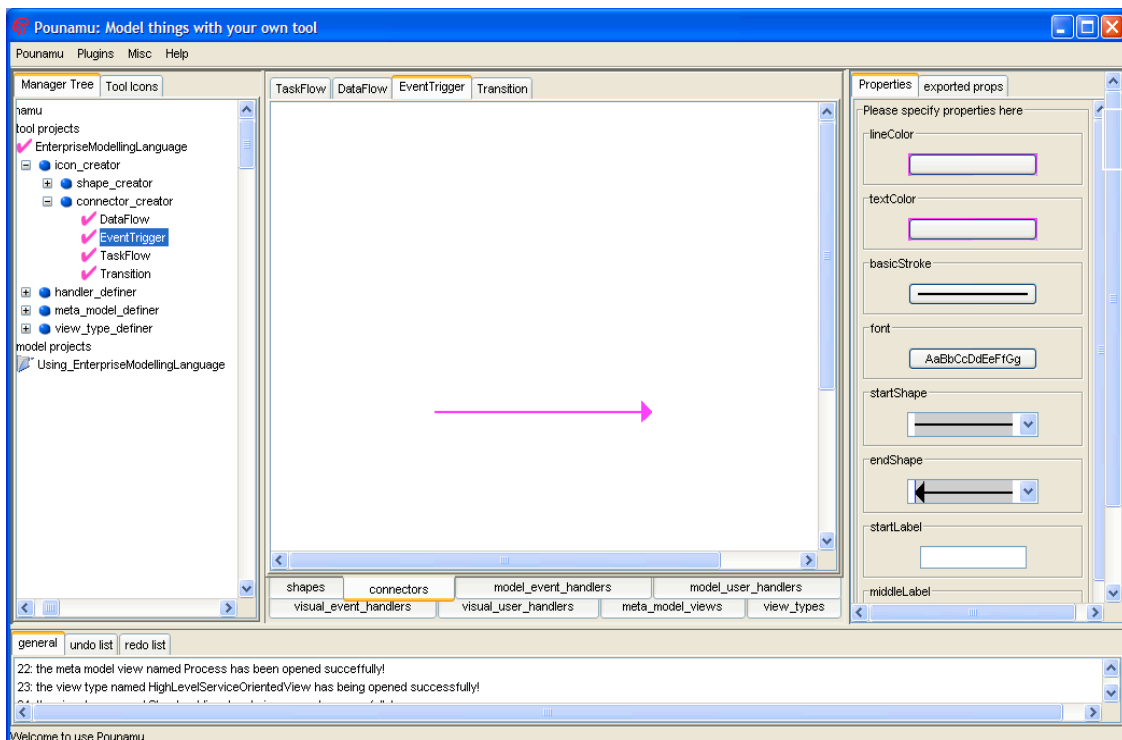
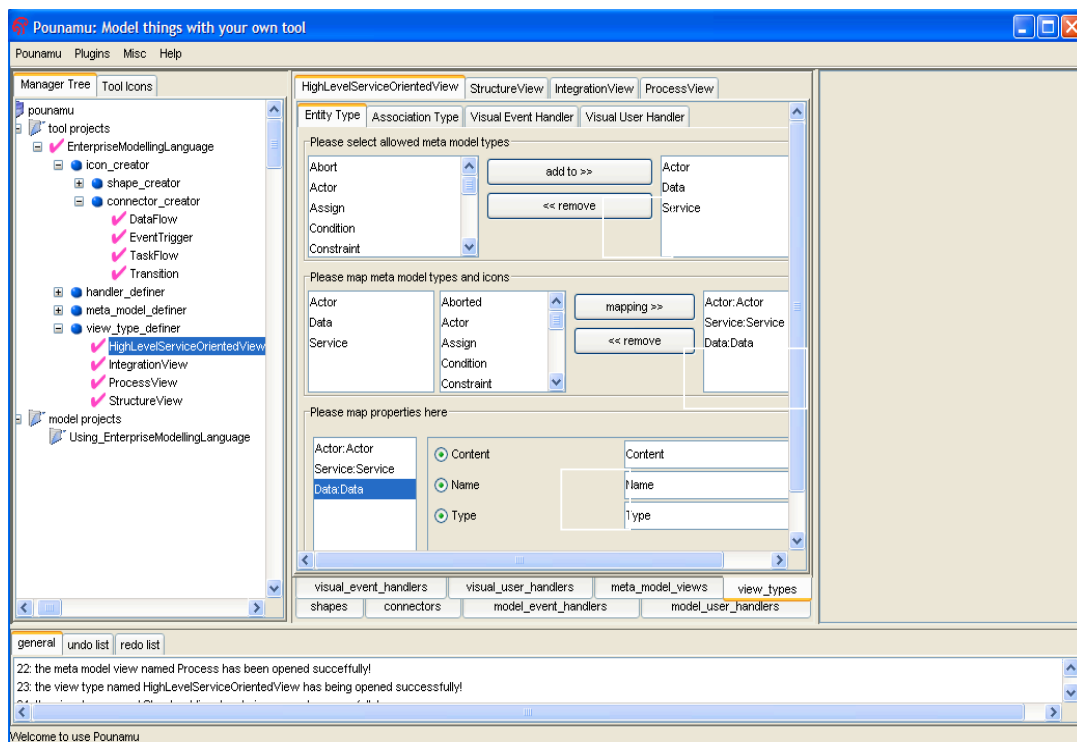


Figure 4.4: Construction of the EML connectors in Pounamu

We then defined different “view types” by bringing together required meta-model elements and visual icons and creating mapping relationships between them. A view

type is a diagram type in Pounamu made up of meta-model entities, associations, shapes, connectors and event handlers (specifying dynamic constraints and editing behaviours). As shown in Figure 4.5, view elements and mappings were added using a form-based diagram. The user normally starts by selecting meta-model entities and associations from (a), and then maps them to visual icons in (b), and finally specifies mappings of model properties<sup>(c)</sup> to visual properties in (c).

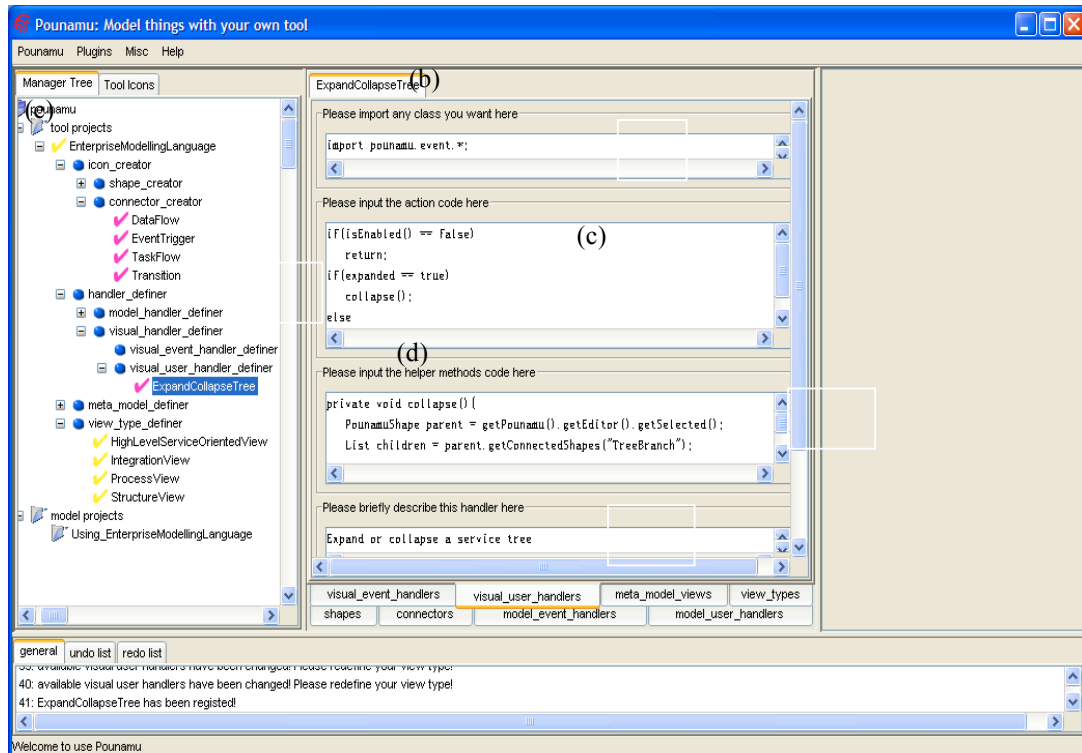


**Figure 4.5: Construction of the EML view types in Pounamu**

We then defined “event handlers” to react to = model or diagram changes. Event handlers in Pounamu provide flexible behaviour specifications for a tool. As shown in Figure 4.6, handler snippets were added in a form-based editor. Compile-time checking of the code was performed when an event handler was registered. To define an event handler, the user needs to select the event handler (ExpandCollapseTree) node from the Manager tree window (e), and then add into the form-based editor import statements (a), action codes (b), helper methods (c) and the description (d).



(a)



**Figure 4.6: Construction of the EML event handlers in Pounamu**

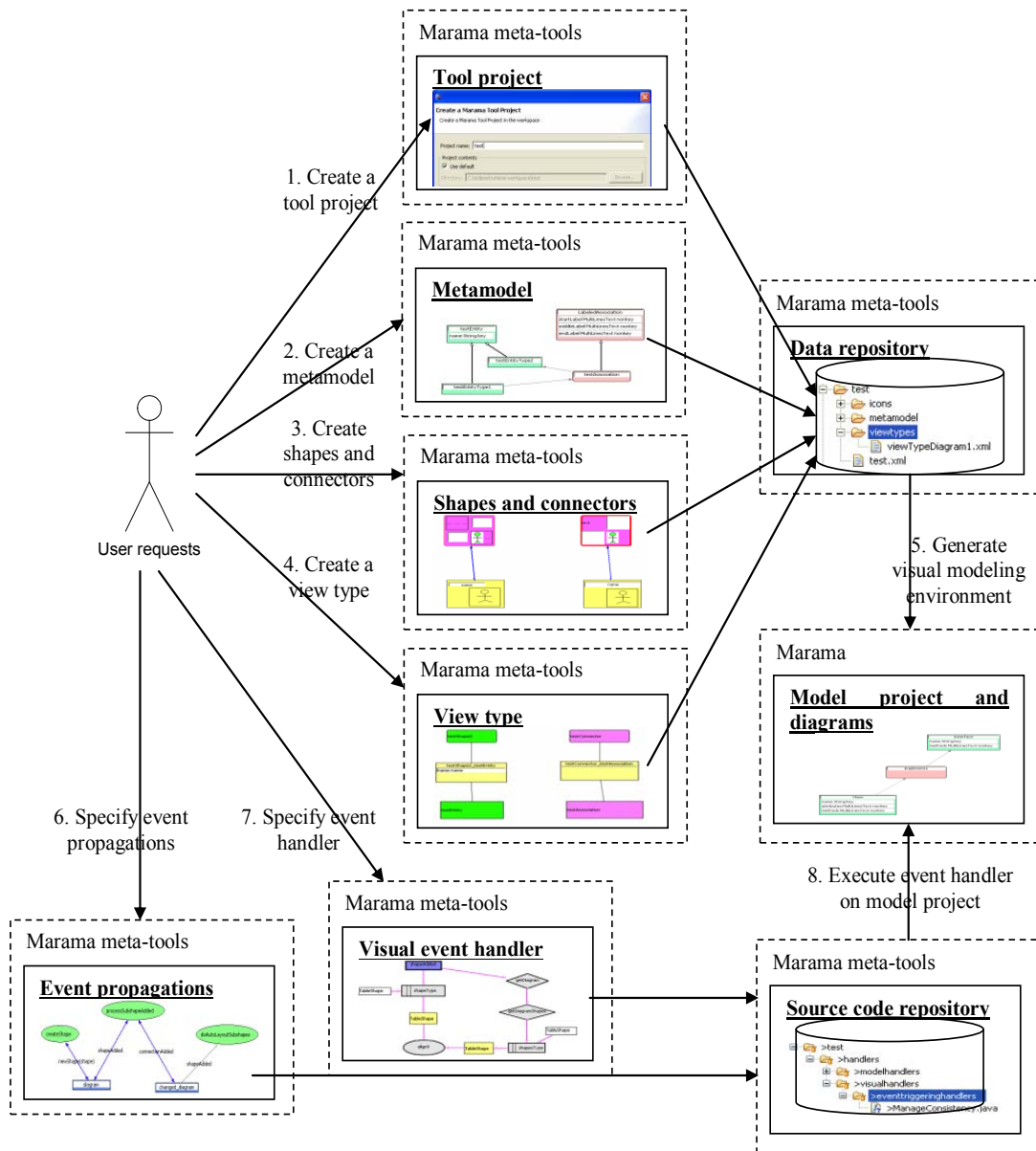
### 4.3 Marama EML Tool

We then rebuilt our Pounamu-based tool using an initial Marama editor prototype in Eclipse. This provided an Eclipse-hosted version of our EML modelling tool using Eclipse views, editors, graphical modelling tools, data management and editing controls.

The final EML prototype we developed was defined using the Marama meta-tools (Grundy and Hosking et al 2006). The structural aspects were constructed visually using the Marama diagramming-based editors. The behavioural aspects were constructed programmatically as extensible Marama event handlers. End user tools can be specified using Marama meta-tools in the following steps (Li and Hosking et al 2009) as shown in Figure 4.7:

- 1) Create a tool project using the Marama Tool Project Wizard.
- 2) Specify the tool metamodel including entity and association types and their attributes, sub-typing relations and OCL constraints.
- 3) Create shapes/connectors to represent metamodel entity/association types.
- 4) Compose a view type by specifying shapes, connectors, entities, associations, view-model mappings and visual constraints
- 5) The initial tool project can be used for defining the structure of end user domain models.
- 6) – 7) Though further event propagations and event handlers can be specified visually using ViTABaL-WS (Grundy and Hosking 1995) and Kaitiaki (Liu and Grundy et al 2005) views, programming with Java using Marama API was necessary due to incomplete development of Marama's visual event handlers.
- 8) The event handlers are inserted into model project instances to take effect.

In the following section, we elaborate in more detail on the implementation of MaramaEML using the Marama meta-toolset.



**Figure 4.7: Tool construction steps using Marama meta-tools (Li 2007)**

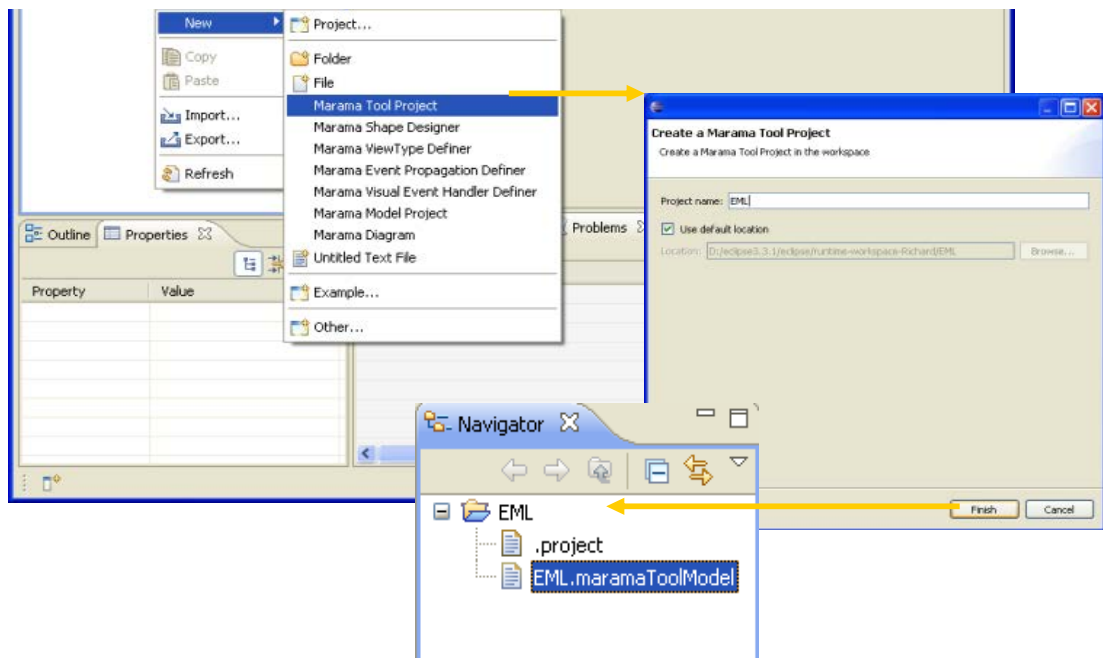
### 4.3.1 Structural Backbone

Marama meta-tools support diagramming-based tool specifications using three definer views: the Metamodel Definer view, the Shape Designer view, and the View Type Definer view. The Metamodel Definer is used to define the metamodel of a tool, including entity types, association types, attributes, constraints and model level event handlers. The meta-model is the underlying data structure of the end user tool. The Shape Designer is used to construct shape and connector icons via drag-and-drop and

property specifications. The shapes and connectors are the iconic representations of the backbone model data. The View Type Definer is used to compose the view elements and the mapping relationships between meta-model elements and the shapes/connectors.

### **4.3.2 Tool Project**

The EML tool was initially created using the Marama Tool Project wizard, as shown in Figure 4.8. The tool project was created with placeholders for the meta-model specification, requiring the definition of the meta-data for the EML language, and then filled in with addition of visual shape designs and view type composition. The EML tool was developed in an iterative way. We could create additional meta-model and iconic elements at any stage once the basic tool project had been established. Where complexities were needed, we defined event handlers on top of the structural backbone using the Marama APIs.



**Figure 4.8: Creating the EML tool using Marama.**

### 4.3.3 Metamodel

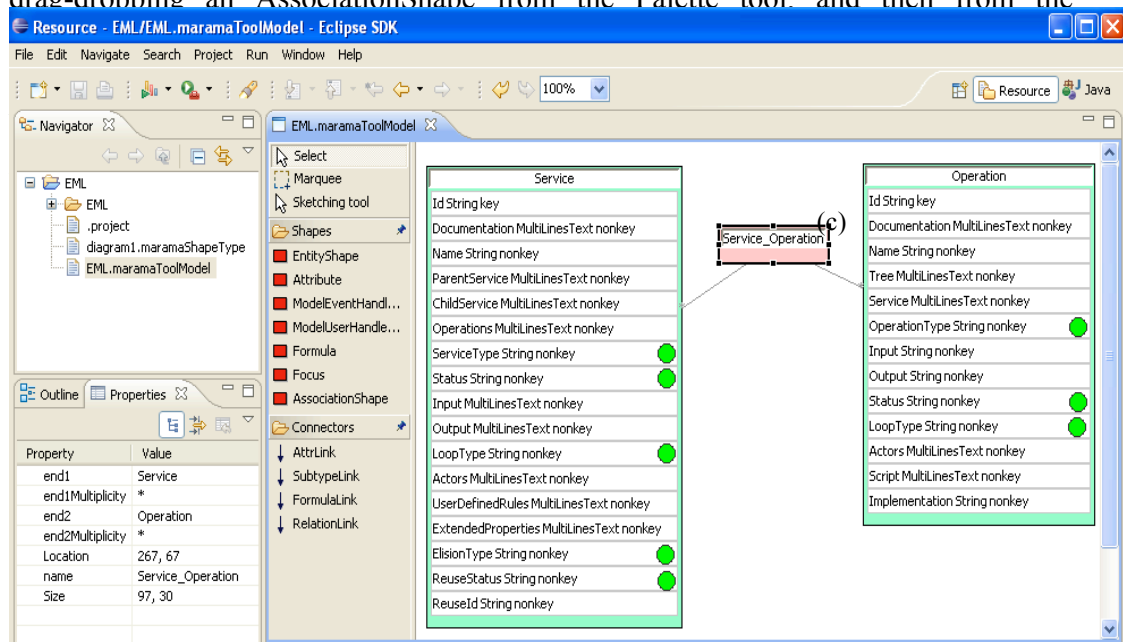
The EML metamodel is an Extended Entity Relationship (EER) model that consists of a range of entity types, with refined attributes and association types. Constraints were added in the form of OCL expressions. Figure 4.9 shows part of the EML metamodel specification that includes the Service and Operation entity types, related by the Service\_Operation association type. In this example, the user defines the service and operation entities and association in (a), and for each of them, they can modify the detailed properties in (b). The green circles in entity shapes are the formulas that are used to describe the derived or preset values of the attributes. Their detailed contexts and OCL expressions are represented in (c).

The entity types (Service and Operation) were created by selecting and drag-dropping

an EntityShape from the Palette tool. The name of the entity type is defined using the Properties view. A number of attributes are placed into an entity type. In Marama, an attribute can be specified as a key, constraining the uniqueness of its value in a model instance; it also has a name and type.

(a)

The association type (Service\_Operation) were created by selecting and drag-dropping an AssociationShape from the Palette tool and then from the



id	context	expression
1	Service.ServiceType	Set{'Service', 'Sub-Service'}
2	Service.Status	Set{'Un-executed', 'Skipped', 'Finished', 'Failed', 'Aborted', 'Other'}
3	Service.LoopType	Set{'None', 'Single Service Loop', 'Two Services Loop', 'More than Two Services Loop'}
4	Service.ElisionType	Set{'Collapset', 'Extend'}
5	Service.ReuseStatus	Set{'False', 'True'}
6	Operation.OperationType	Set{'Receive', 'Send', 'User', 'Script', 'Abstract', 'Manual', 'Reference', 'None'}
7	Operation.Status	Set{'Un-executed', 'Skipped', 'Finished', 'Failed', 'Aborted', 'Other'}
8	Operation.LoopType	Set{'None', 'Single Operation Loop', 'Two operations Loop', 'More than Two operations Loop'}

Figure 4.9: Defining EML metamodel in Marama.

A variety of OCL constraints were added into the EML metamodel. They were represented in the form of formulae, the green circle nodes in the diagram. Each formula has its context (the entity/association or attribute where the constraint is residing) and expression (the OCL expression). The Formula View at the bottom of Figure 4.9 lists the formulae placed on various attributes of the Service and Operation entity types. The common ‘Set{ }’ expression constrains an attribute to a list of predefined values. The full EML metamodel is described in Chapter 3 (Enterprise Modelling Language).

We have later integrated a BPMN and a Form Chart view to allow their notations to coordinate with EML in MaramaEML. For such integration, we had to add relevant BPMN and Form Chart meta-model elements (e.g. the process, task, object and sequence flow notations of BPMN; the page, action and transition notations of the Form Chart) into the existing EML meta-model. The construction process of those meta-model elements is the same as what we explained above.

#### **4.3.4 Shapes and Connectors**

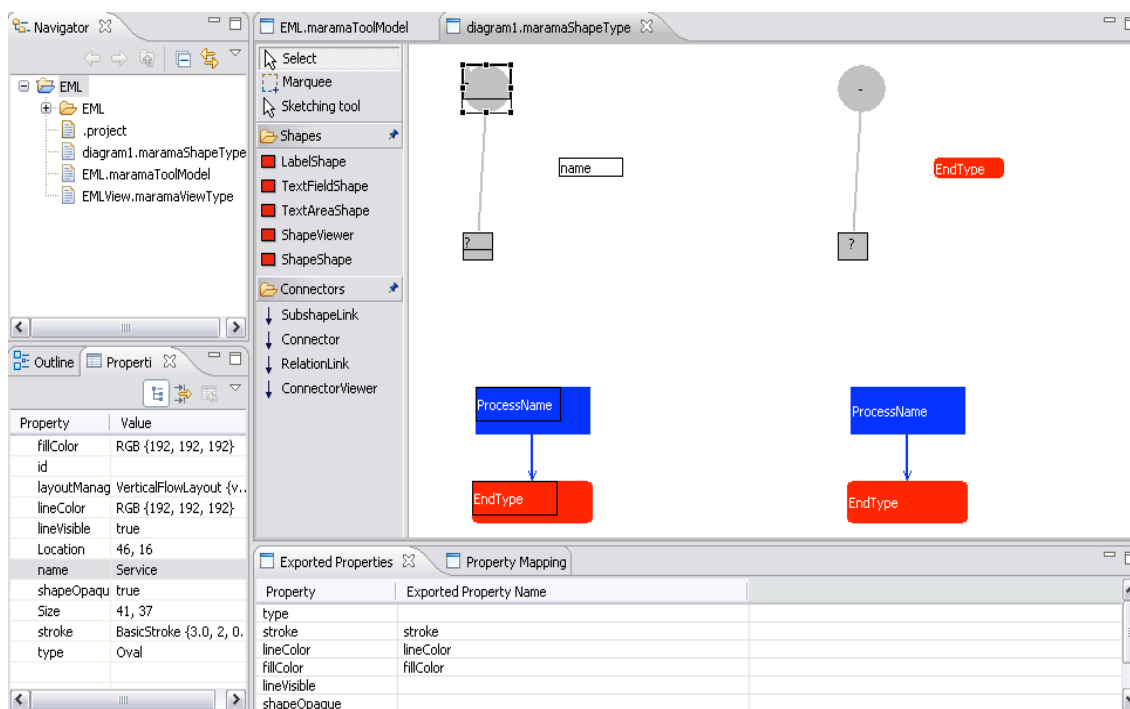
The visual representations of the EML meta-model elements (and those of the BPMN and Form Chart elements) were designed using Marama shapes and connectors. Figure 4.10 shows a Shape Designer view that contains several shapes (Service, Operation, Name, ProcessStart and ProcessEnd) and connectors (ProcessFlow and TreeBranch).

The shapes were created by selecting and drag-dropping a ShapeShape from the Palette tool first as the base container, and then fill in with labels, text fields or text areas. Various visual properties including the colour, layout and display text were set via the Properties view.

The connectors were created in a similar manner, but needed to connect two shapes.

Each shape/connector in design has an accompanied concrete viewer to the right hand side. The viewers show immediately the runtime effect of the design, i.e. the exact representation that will appear in the end users' models.

Various properties were exported using the Exported Properties view. To export a property, we just need to define an exported name for an existing shape/connector property. The purpose of exporting a property is to allow end users to modify it at runtime for a model instance, and allow their mapping to meta-model properties. Non exported properties are hidden from both the user and the underlying model.

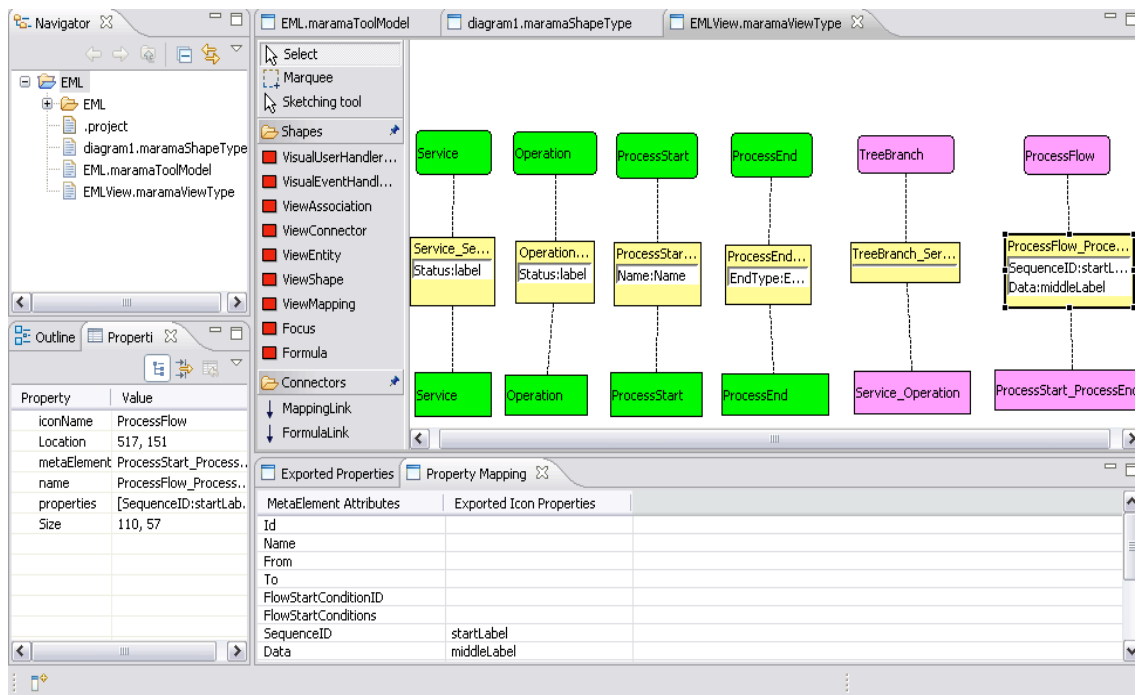


**Figure 4.10: Defining shapes in Marama.**

### 4.3.5 View Types



A Marama view type is used to specify the composition of a view (shapes and connectors) and the mapping from meta-model elements to those shapes and connectors. The one-to-one mapping (yellow rectangle) of a shape (rounded green rectangle) to an entity type (green rectangle) and that of a connector (rounded pink rectangle) to an association type (pink rectangle), together with all of the corresponding property mappings were defined to compose the EML view type as shown in Figure 4.11. There we see, for example, the Service shape mapped to the Service entity type, the Operation shape mapped to the Operation entity type, and the ProcessFlow connector mapped to the ProcessStart\_ProcessEnd association.



**Figure 4.11: Defining the view type in Marama**

An entity/association type was added by selecting and drag-dropping a ViewEntity/ViewAssociation from the Palette to the diagram, and then the name of an entity/association type was selected from the Properties window. We were able to do this because Marama queried the user-defined metamodel and loaded the available elements into the property sheet.

A shape/connector was added in the similar way, by selecting and drag-dropping a ViewShape/ViewConnection from the Palette to the diagram followed by selecting the name of the shape from the Properties window.

A mapping was added by selecting and drag-dropping a ViewMapping from the Palette to the diagram first, followed by selecting a shape/connector and an entity/association type from the Properties window. Mapping links were automatically generated into the diagram to connect the mapped elements. Once a mapping relationship was set up, further mapping of properties between the mapped elements were selected via the PropertyMapping view (shown in the bottom window of Figure 4.11).

By using the Marama framework, multiple linked views associated with an underlying meta-model can be created. The multiple view technology allows multiple visual language notations to be integrated to be used for flexible and interchangeable model specifications. We wanted to provide EML with the ability to allow users to choose to use their preferred notation to model their systems flexibly, i.e. using the BPMN and Form Chart specifications, so we provided their linked notational views in MaramaEML. The same implementation as described above has been carried out to define the BPMN and Form Chart views with their relevant meta-model elements, shapes and connectors, and mappings in between.

#### **4.3.6 Model Projects**

The above EML tool definitions allowed Marama to automatically generate an EML modelling environment with structural modelling capabilities. End users can now start modelling (i.e. creating EML model instances) using the EML notation in the generated environment.

A model project needs to be created to package an end user model. Marama provides

a wizard for creating model projects in an easy way, allowing the end user to select an available tool project so that the modelling environment based on that tool can be generated.

Once the model project is set up, the end user can create diagrams using the Marama Diagram wizard, and then create domain models by selecting and drag-dropping the EML elements from the Palette into the diagram, followed by domain property settings.

While constructing the structural backbone of the EML tool was a simple visual experience using Marama, adding dynamic behaviours for the EML tool was a non-trivial task. Marama does support visual behaviour specification in various ways (Liu 2007), but those techniques were not fully available at the stage of the EML tool development. We had to resort to code to implement behaviours such as the automatic Tree Structure layout, showing/hiding Process Overlay and code generation.

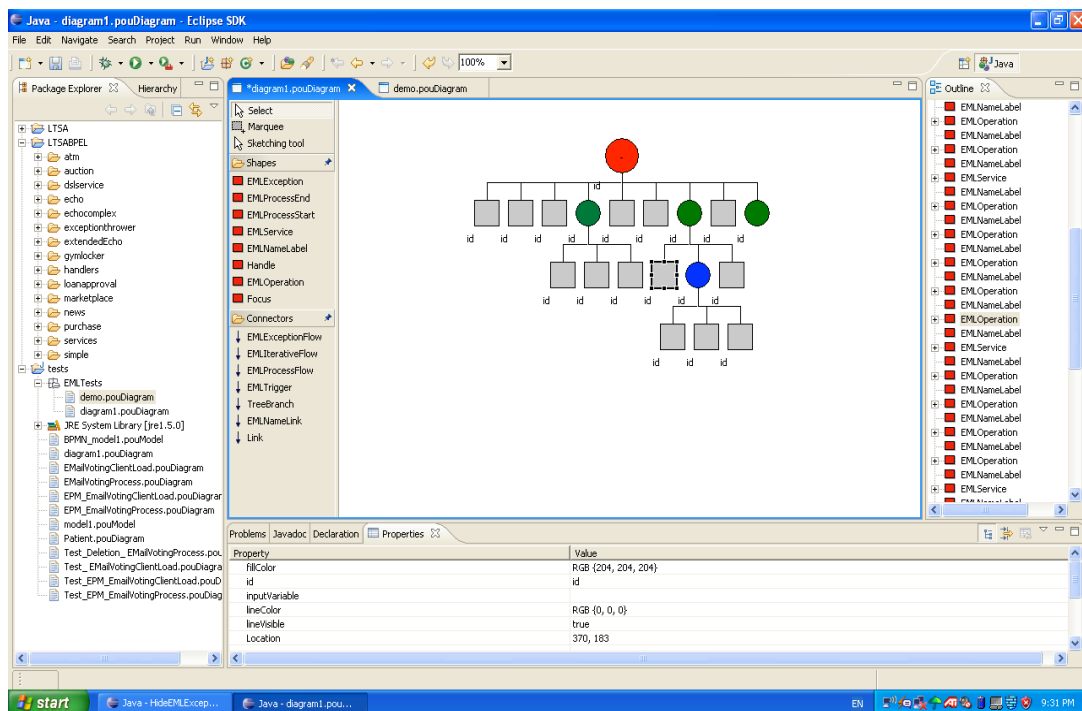
#### **4.3.7 Behaviours**

Marama provides extension points for tool behaviour specifications. These extension points are known as event handlers. Marama uses the EMF notification mechanisms to generate events for any change in its models and diagrams. Tool developers can use the Marama APIs to catch or filter the generated events. User defined procedures can handle various built-in event handler types, such as the new shape/connector added event, shape/connector resize/move/delete event, entity/relationship property change event, and the user action event type, i.e. the right click action on context menus.

#### **4.3.8 Service Tree Structure**

An event handler for the EML tree layout was defined for the root/leaf and parent/child shapes to respond to. When a shape is added to an EML modelling view,

the location of the shape is analysed and then corresponding reacting behaviours are executed to automatically layout the shape based on whether the shape is created as a child of a parent shape or is standalone. This event handler catches the new shape added event and responds with a tree layout algorithm. We have developed a algorithm to calculate the vertical and horizontal space between all the nodes in this tree structure. When the user adds a new service or operation node under a service node, the positions of all previous nodes are recalculated and updated to maintain the tree layout; tree branches are rebuilt too.. All the name spaces that belong to the nodes are automatically moved as well. Figure 4.12 shows an exemplar EML tree structure in MaramaEML.



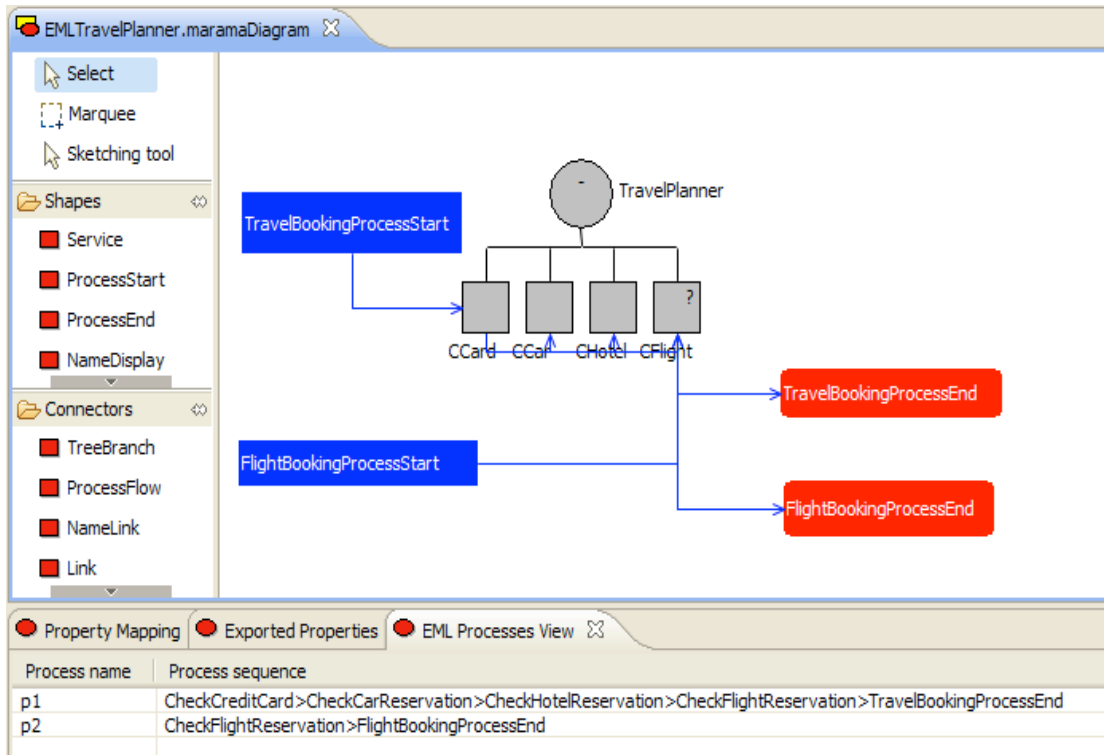
**Figure 4.12: the Tree layout in Marama\_EML**

A sub-tree can be moved to a standalone location to become an independent tree or to a parent shape location to become a migrated child. This is implemented as an event handler to react to a shape move event. When a shape is moved, all its subsequent descendants are retrieved and moved together as a whole when reacting to the event.

### 4.3.9 Overlay

The overlays of process models on a tree structure were firstly defined as normal elements, including the Process Starts, Process Flows and Process Ends as the basis, with Trigger and Exception Flows as complements. Iterations can be defined for each Process/Trigger/Exception Flow via iterative property settings. Multiple process overlays can be modelled in the same diagram, and they are distinguished from one another using unique process identifiers (each process-related element has an ID property showing which process it is belonging to). Process overlays can be shown or hidden, selected and deleted, to react to the user's interaction. These were implemented as user triggered event handlers (reacting to right click actions on context menus). The context menus added by implementing the process overlay event handlers include the *show/hide all process overlays*, *delete all process overlays*, *select/show process [name] while hiding other processes*, and *delete process [name]*. In addition, an Eclipse ViewPart implementation called "EML Processes View" provides the user with a more straightforward way (juxtaposed display with the diagram, as shown in Figure 4.13) to view all the processes overlaid in a diagram, also allowing selection of a particular or a subset of processes to display in the diagram.

In the implementation of process overlays, all the elements of an EML diagram are walked through to identify their notation type and properties. Elements related to process overlays are collected and distinguished using a Hashtable data structure, where they are traversed through and analysed to supply on-demand interactive display of multiple process overlays.



**Figure 4.13: Process overlays in EML**

#### 4.3.10 Code Generation

Saving an EML model from the Eclipse workbench or clicking a context menu called *generate BPEL from EML* will both generate BPEL code to a user’s environment. The code generation facility was implemented as both embedded runtime behaviour in Marama (by adding to the Marama API) and a user triggered event handler. It uses the Marama API calls to query user-defined modelling elements and perform mapping code generation to the file system.

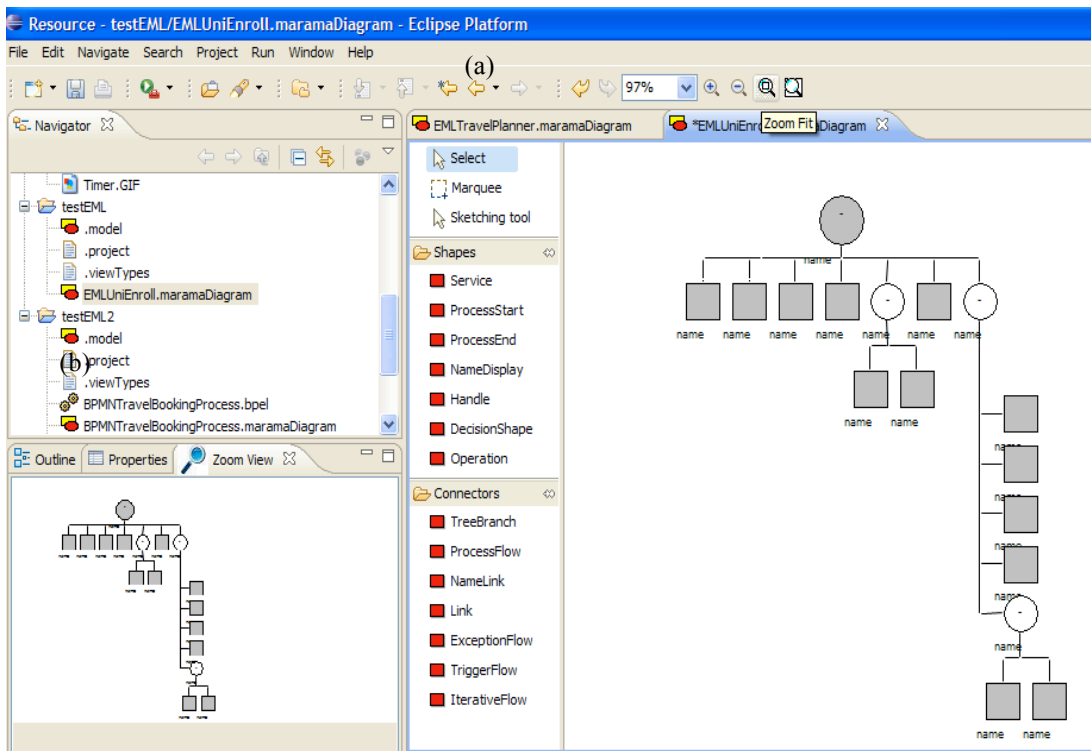
The algorithm used for generating BPEL is straightforward, containing a traversal of the EML nodes using a Hashtable data structure and analysing the types and properties of the EML elements to permit mapping to the corresponding BPEL code structure. As multiple processes can be defined in one EML diagram via process overlays, multiple BPEL process files can be generated. Basic one-to-one mapping of EML elements to BPEL constructs is achievable, for instance, an EML Service maps

to a BPEL PortType; an EML Operation maps to a BPEL Operation; an EML ProcessFlow maps to a BPEL Link; an EML ExceptionFlow maps to a BPEL Compensation Handler; an EML TriggerFlow maps to a BPEL Event Handler. The code generator buffers the diagram analysis results, i.e. the XML code snippets contributing to the final BPEL processes definition, and finally outputs the XML files. However, generating complete and executable BPEL code requires additional diagram properties (e.g. input and output data, conditions, error message etc.) to be present over and above the basic EML modeller. This required us to re-engineer the EML language prototype to add these properties to EML elements.

A trial generation of BPEL from EML's complementary BPMN views has also been implemented using MaramaTorua (Huh and Grundy et al 2007), which is a locally developed visual mapping tool allowing user-defined mappings from one language schema to another. The trial was successful, demonstrating the ease of code generation by visual specification via the mapper without the need of a backend code generator from EML. Our next step is to explore a similar mechanism in generating BPEL from EML with the hypothesis that the process will be equally straightforward.

#### **4.3.11 Zoomable View**

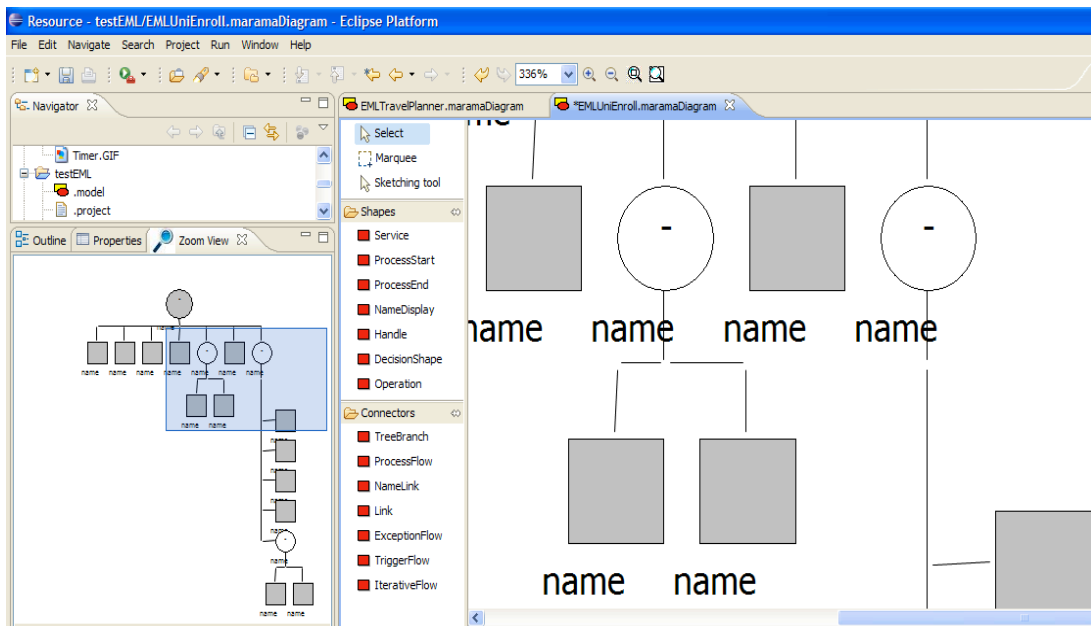
Zooming functions (Singh and Mitra et al 2004), including zoom in, zoom out, zoom fit and selection zoom are implemented as both toolbar commands on the Eclipse Workbench (a) (as seen on the top part of Figure 4.14). In addition, we have added an Eclipse PageBookView, which listens to user's diagram selection events, and renders a 'Radar' zoom view (b) for the whole EML diagram accordingly (as seen at the bottom left part of Figure 4.14).



**Figure 4.14: Zooming commands and zoom view**

The “zoom in” function zooms in the entire EML diagram by a predefined scaling factor. The “zoom out” function zooms out the entire EML diagram by the same scaling factor. The “zoom fit” function provides the very best view of the EML diagram fitting all elements in the available screen space with an automatically adjusted scaling factor. The “selection zoom” function allows user to select a square area of the diagram and zoom into the selected part. The “Radar” zoom view accompanies the EML diagram, providing a thumbnail as well as an indication of visible items inside the screen boundary and those outside of the boundary of the EML diagram. As shown in Figure 4.15, while the EML diagram is zoomed in showing the selected elements as the focus, the “Radar” zoom view also indicates the selection boundary. Continuous zooming based on an existing zooming status is also allowed.





**Figure 4.15: Selection zoom**

This implementation is integrated with the Marama API. We have provided an additional package inside the MaramaEditor plug-in to manage the zooming functions while still exploiting the existing Marama code base. The package includes zooming interfaces, various zooming actions, Marama diagram mouse trackers, and viewing areas. MaramaEditor is then configured to enable these zooming features using its zoom manager.

### 4.3.12 Fisheye View

The fisheye view function (Gansner and Yehuda et al 2004) in MaramaEML provides a way to render a small focused display of a large EML tree structure. While the amount of information created by the user increases, the viewing space of MaramaEML remains relatively small and thus has a limitation. The idea is to present a large amount of EML diagram data to users in a way that is searchable, and getting information is not too timely of a task.

We implemented a fisheye view function by providing a local context against a global context. This is a focus and context visual technique which can often be referred to as

a “distortion based display”.

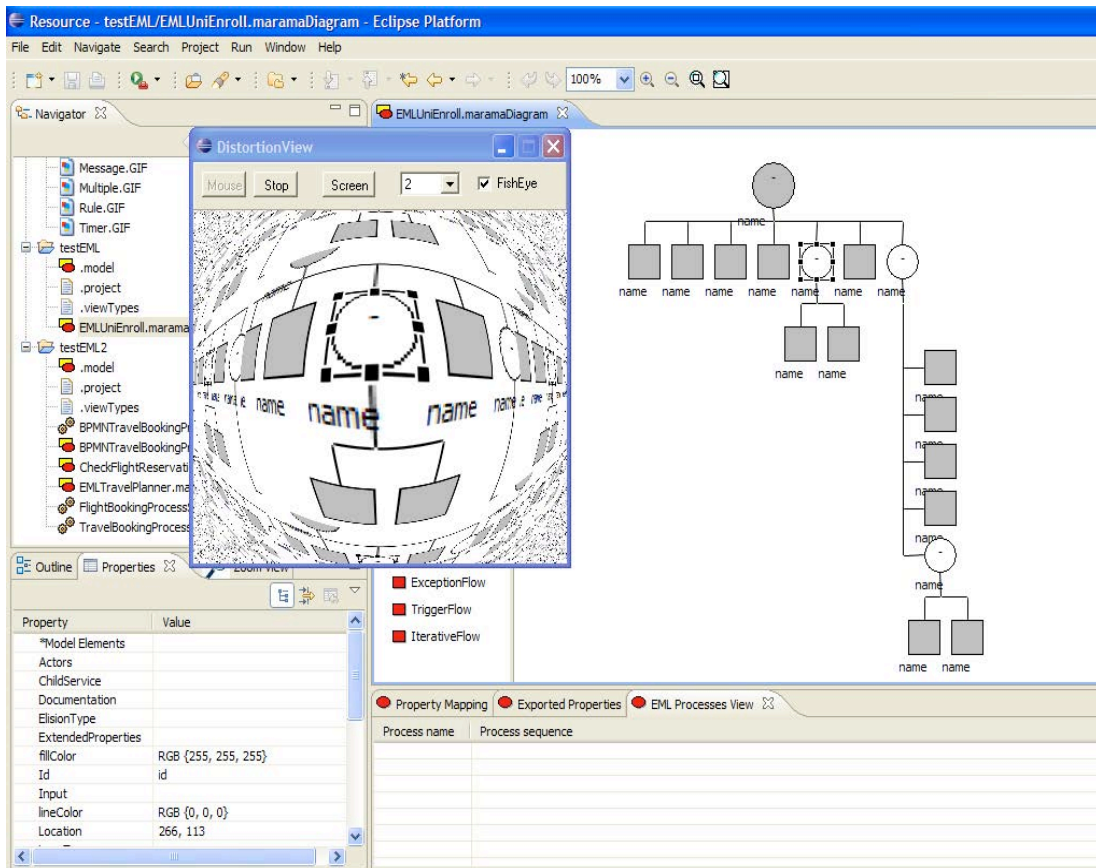
Three major attributes have been developed in our implementation to achieve the fisheye function: Focal point, Distance from focus ( $D(x)$ ) and Degree of Interest (importance, resolution:  $DOI(x)$ ).

A point of interest has to be defined so that the interaction with the focus, meaning what is going to be the global context, can be determined. The “distance from focus” concept determines the distance from my point of interest (focus) to some point  $x$ . Examples of “Distance from focus” could be the distance from the centre of a service node or operation node to the centre of another service or operation node, or from a root node directory to the lower level of leaves node on the EML tree structure. Longer distances lead to smaller sizes of the shapes.

Degree of Interest (DOI) is another concept in the fisheye view implementation. For a user at any given point of interaction within a system, he/she is not going to be interested in the entire system all of the time. For a particular purpose, it is necessary to determine how interested a user is in an application on the system. As a result, DOI would help the software to represent parts of the EML tree structure that are of most interest to the user in great detail, while the other parts that will not be used often would be in less detail. A higher degree of interest is indicated by a higher value.

The implementation of DOI is composed of a static component and the dynamic component. The static component is either the priori importance or the global importance of the element relative to every other object in the system. For the user, the global importance is how a tree node is used more than another tree node in the EML diagram. The dynamic component creates a relationship between the user’s interest and the importance of an item depending on the latest interactions on the tree. The DOI is assigned to every element in the EML diagram, and a node is selected as the central focus point. It is important to notice that if the point of interest changes,

then the DOI must be recalculated for every node.



**Figure 4.16: Fisheye view of a Diagram in MaramaEML**

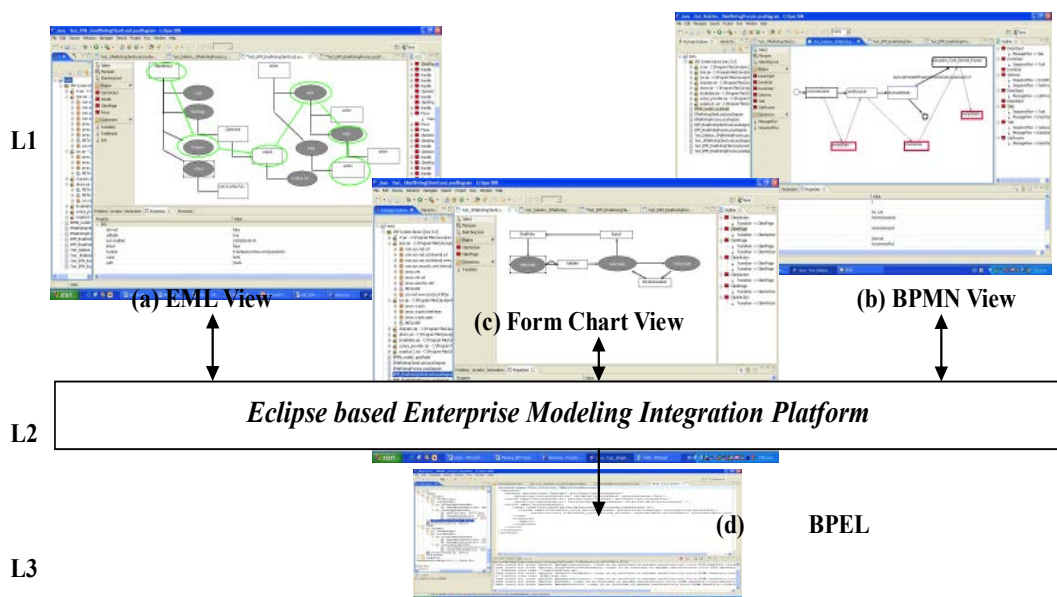
Figure 4.16 shows a fisheye view example based on the EML tree structure. The mouse pointer is the default Focal Point, the DOI of the certain part of the tree structure is based on the Distance of Focus. A shorter distance will lead to higher value of DOI, thus, the shape will be represented as a larger size. The longer distance brings a lower value of DOI, which leads to smaller sized shapes. As the mouse moves, the DOI value and shape size of the tree nodes are changed dynamically. The fisheye function has also been applied to the process, trigger and exception overlays of the EML trees and BPMN diagrams in the MaramaEML environment.

## 4.4 Integration

The Eclipse DOM XML parsing APIs were primarily used for integrating EML with other modelling technologies. As Marama generates both XML and XMI backend for model and diagram interpretation, and it provides APIs for parsing XML in an easy way, we were able to easily integrate the EML tool and its generated user models with a relatively low amount of effort. We have developed an integrated support tool for EML to supplement its functionality with other mainstream notations.

**Figure 4.17: EML Integrated Tool Framework**

Figure 4.17 shows an overview of the integrated framework. It allows the user to construct and manage EML (a), BPMN (b) and Form-Chart (c) diagrams and automatically generates executable BPEL code (d). It thus builds a strong relationship with industry business process modelling standards.



We define three layers for this multi-layer framework; visual (L1), tool (L2) and code (L3) layers. In general, it is sufficient for the user to use the visual layer to model enterprise processes. By using corresponding schema, the tool layer facilitates

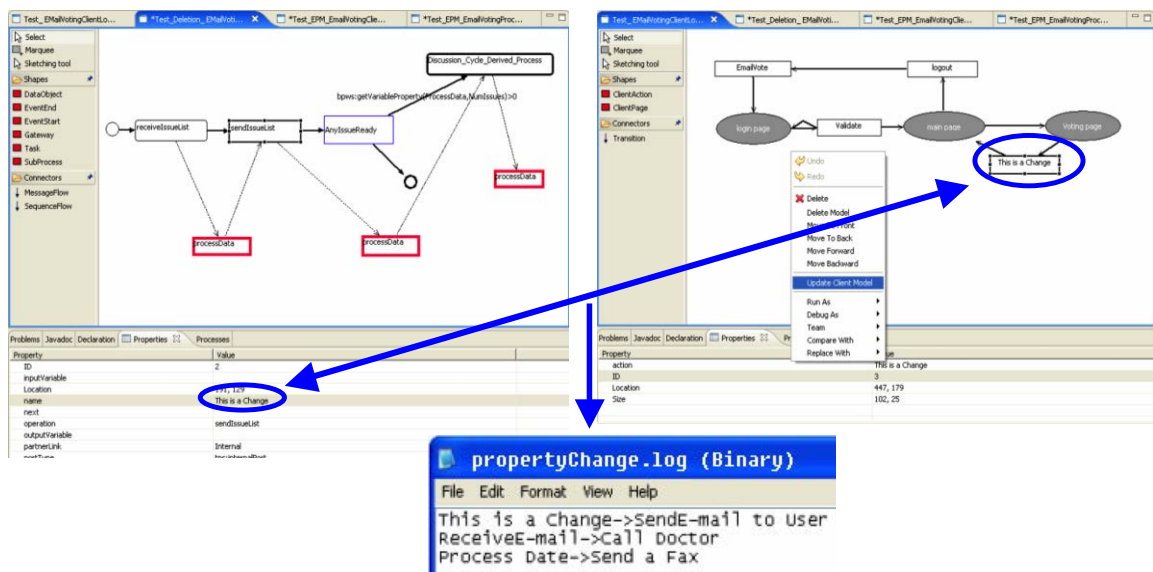
automatic code generation and maps between the visual notations of level 1. Figure 4.17 shows an example of mapping between EML, Form Chart and BPMN that is achieved through BPEL code at the back-end.

This framework provides a good launch pad to enhance the integration and generation ability of different notations. By using the XML-based BPEL code as a middleware, a single notation can be integrated effectively with other modelling technologies. It provides users with a real multi-view function for enterprises, as they can have views based on different notations, and all the views are automatically kept consistent. This integration approach provides multi-level framework support for flexible and broad integration of complex enterprise system models.

Model-View-Controller is the underlying pattern that we have adopted for implementing this EML integrated tool (Buschmann and Meunier et al 1996).

**Figure 4.18: Consistency Mapping Between a BPMN view and a Form Chart**

Semantic consistency of the three views is implemented using Java event handlers. Model views are checked on-the-fly for consistency violations after each model modification. There are primary underlying mappings defined in this software as crossovers between the EML and BPMN server-side specification, and also between client-side service calls and server-side definitions. Event logs are kept and used as



resources to trace model changes and corresponding consistency updates of the three views.

Figure 4.18 demonstrates a run time automatic mapping example between BPMN (a) and Form-Chart views (b). We have used BPMN to model a server side process of an E-mail voting system and a Form-Chart to model the client side. When a user changes the property name from BPMN view (“SendE-mail to User”), they can use the “Update Client Model” function to build the mapping to the Form-Chart view automatically. All changes they have made are automatically recorded in a “propertyChange.log” file (c).

A locally developed mapping tool VMLPlus, has also been used to specify complex mappings between the EML and BPMN notation. Users can select and link elements from visual schemas of the two notations. This allows deeper level mappings to be specified facilitating more complete consistency updates, including types and attributes.

Performance simulation (Grundy and Hosking et al 2006) is also incorporated in the integrated EML support environment, facilitating cost-effective tests of the integrated specifications using random data and visualisation of test results using the same design-level specification views.

## **4.5 Summary**

In this chapter, we have discussed a set of implementation issues for MaramaEML and its early prototype, Pounamu version of the software. The implementation of EML has been an iterative process, with continuous support of progressing meta-tools. Though we needed to migrate EML from Pounamu to the Marama, this did not take up a lot of effort, as the underlying themes of Pounamu and Marama have major commonalities. The constructions of the structure backbone of EML were easy and

efficient in both Pounamu and Marama, however, those of the behaviour extensions required massive amount of work. In order to add onto Marama and define end user tool interactions, we had to exploit the Marama APIs which requires well-established and detailed knowledge of the underlying framework infrastructure. The Zooming and Fisheye view functions have been developed as a plug-in for Marama to extend the scalability and usability of MaramaEML.

# Chapter 5

## CASE STUDY

---

We were asked to model a large university enrolment system (including sixteen services and fifty six functions) as a part of a process improvement exercise. In this chapter, we use this complicated example to demonstrate the main modeling capabilities of EML and various support functionalities provided by the EML environment. In section 5.1, the enrollment system is briefly introduced. The service tree modeling example is described in section 5.2. Section 5.3 covers the multi-overlays structure for processes, exceptions and triggers. The multi-view support function is discussed in section 5.4, and the automatic code generation and validation are presented in section 5.5. Section 5.6 describes the zoomable and fisheye view support in the MaramaEML environment. The final code deployment is reported in section 5.7.

### 5.1 University Enrollment System Example

The university enrolment system is a complex enterprise system that involves dynamic collaborations among five distinguished parties: Student, Enrolment Office, Department, Finance Office and StudyLink (the New Zealand government's student loan agency).

The main functional requirements are:

- Students will use this system to search the course database and apply for enrolment in target courses; if their application is approved, they may want to apply for a loan from StudyLink;
- After receiving student applications, the Enrolment Office checks the academic conditions with academic Department staff and then informs Students of the results;



- Department staff check the course enrollment conditions and make the final decision (approve or reject);
- For an approved enrolment application, the Finance Office tracks fee payment and informs the Enrolment Office and Department of any changes. If a Student applies for a loan, the Finance Office also needs to confirm the student information with StudyLink.
- StudyLink investigates the student information with the university and then approves (or declines) the loan application.

## 5.2 Service Tree Modeling

The system decomposition process focuses on how to break down the system to identify its structure and behaviour. Once we identify the domain specific structure and behaviour, the modelling grammar (e.g. BPMN, EML etc.) can be applied to represent the analysis and design concepts. We have reviewed prior system decomposition criteria and models. The following four principles have been adapted to our decomposition process:

- **Minimality:** For every subsystem at every level in the overall structure of the system, we try to keep the redundant state at the lowest possible level. All the states must be reachable.
- **Determinism:** For every event at every level in the overall structure of the system, we only model it either as an external event or a well-defined internal event. When states that lead to two or more post-states, the guard conditions need to be considered (to specify the appropriate path).
- **Losslessness:** Hereditary and emergent states are preserved in the decomposition. The inferences must not be lost when breaking a system into several jointed subsystems.

- Weak Coupling and Strong Cohesion: Models should have minimal external interactions and high internal integration.

Figure 5.1 shows a complex, fully-expanded overview of an EML tree modelling the university enrolment service. The student service, university service, and StudyLink are sub-services (represented as ovals) of the university enrolment service. The university service includes five embedded services (enrolment office, finance office, credit check, department and communication). The rectangle shapes represent atomic operations inside the service. The StudyLink service also includes a detailed four layer sub-service structure.

There are six major functions in Student Service (*Search Course Database, Apply Enrollment, Apply Loan, Make Payment, Modify Enrollment and Receive Information*) and six in the Enrolment Office (*Receive Application, Check Academic Records, Approve Application, Reject Application, Modify Application and Check Other Conditions*). The Finance Office has five direct functions (*Request Payment, Receive Payment, Modify Payment, Send Invoice and Confirm with StudyLink*) and five indirect functions (*Verify Student, Credit Check, Update Information, Scholarship Pay Back and Inform Changes*) via its sub-service (*Credit Check*).

StudyLink owns four different levels of direct and indirect services and operations. More specific details are:

Level 1 (Direct Services):

- Loan Approval Service (includes another three levels of sub-services and six direct operations - *Check University Payment, Approve Loan, Decline Loan, Update Amount, Send Payment and Inform User*)
- Loan Payback Service (has six operations - *Check Amount, Receive Payment, Setup Monthly Payment, Update Information and Send Receipt*)
- Communication Service (has four reusable operations - *Print, E-mail, Phone and Fax*)

Level 2 (Indirect Services & Operations):

- Student Account Management Service (this service is embedded in the Loan Approval Service; it has two sub-services and seven direct operations - *Create Account, Modify Information, Delete Account, Append Account, IRD Check, Income Information and Inform Changes*)

Level 3 (Indirect Services & Operations):

- Interest Calculation Service (this service is embedded in the Student Account Management Service; it has one sub-service and four direct operations - *Add Interest, Reduce Interest, Inform Student and Special Rate*)

Level 4 (Indirect Services & Operations):

- Interest Free Approve Service (this service is embedded in the Interest Calculation Service; it has five direct operations - *Check Student State, Approve Interest Free, Decline Interest Free, Update Loan DB and Inform Other Department*)

EML supports service reuse to reduce structure complexity and increase modelling efficiency. A reusable component is represented in a separate tree. The user pre-defines its structure and saves it in a library. Reusable components have a unique name for future usage. The user can easily attach a reusable component to any branch of an EML tree. In this figure (Figure 5.1), we define the *Communication Service* as a reusable component (at the left bottom), reused by the *University Service* and *StudyLink Service*.

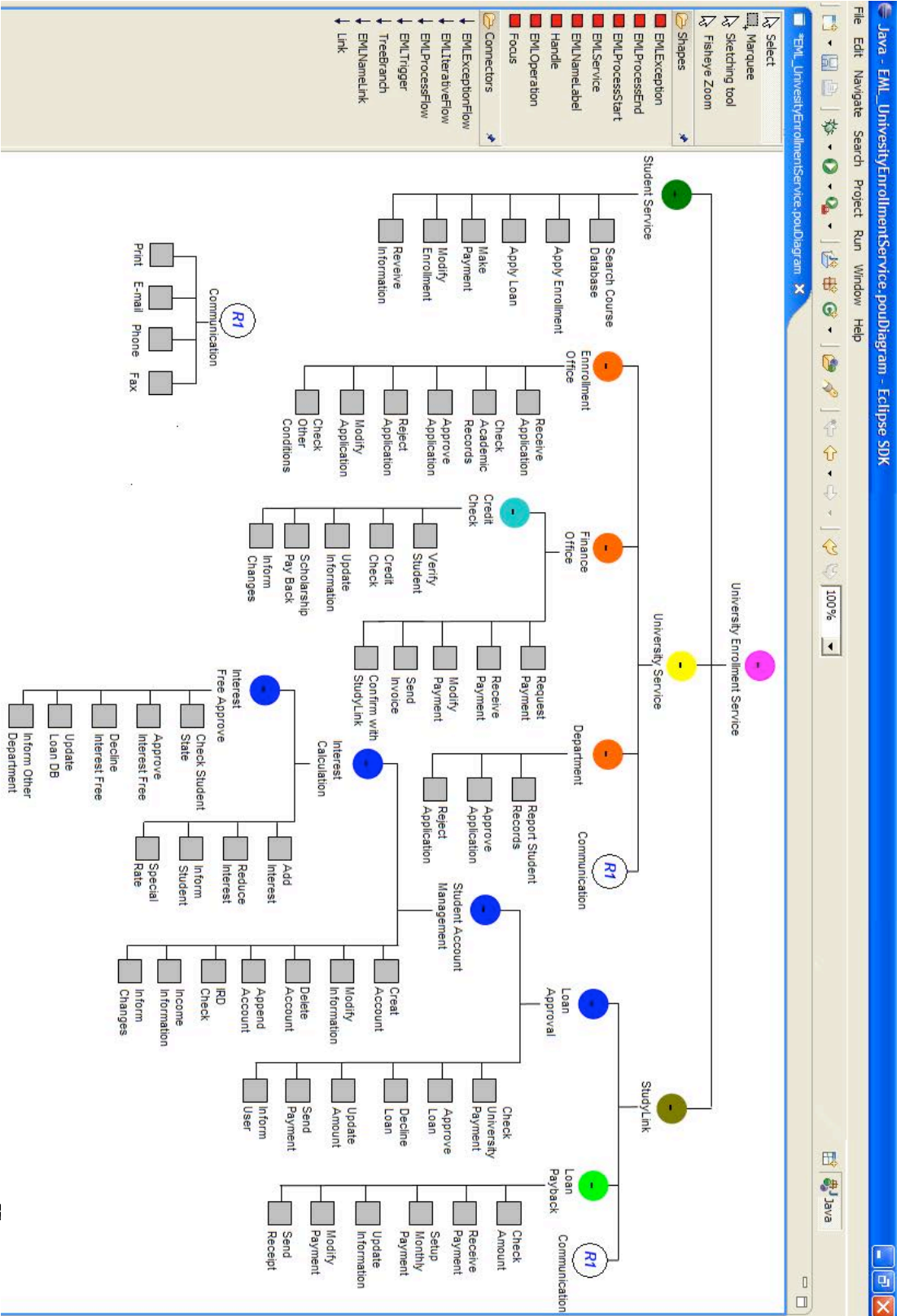


Fig.

Even in this complex model the EML diagram still provides a clear structural view. In an EML-modelled enterprise system, major services are represented as separate trees. In order to mitigate the complexity of the diagram, we use symbols inside each service to identify the elision level of the service visualisation. A minus (-) symbol indicates all activities in the service have been expanded (e.g. all the services in Figure 5.1). A plus (+) symbol indicates that part or all of the sub-tasks (services) are elided (e.g. the *Loan Payback service*, *Student Account Management service* and *Credit Check Service* in Figure 5.2). Every notation in the diagram can be elided and expanded to give users freedom to control the diagram size and complexity. Each tree element has a set of detailed properties e.g. service type, status, input, output, loop, condition, and rule etc.

### 5.3 Overlay for Processes, Exceptions and Triggers

A fundamental part of business process modelling is the representation of flow between stages. In EML each business process is represented as an overlay on the basic tree structure or an orchestration between different service trees. In a process layer, users have the choice to display a single process or collaboration of multiple processes. The user can select Show/Hide EML Process ((a) in Figure 5.2)/ Exception ((b) in Figure 5.2)/ Trigger ((c) in Figure 5.2) Flow functions to view or hide overlays. When a Show/Hide Flow function is selected, a detailed flow list is brought to the screen for further selection. By double clicking the process names in the list, the user can choose to view one (or more) appointed process or all of them. Similar operations apply to the Exception and Trigger Flows.

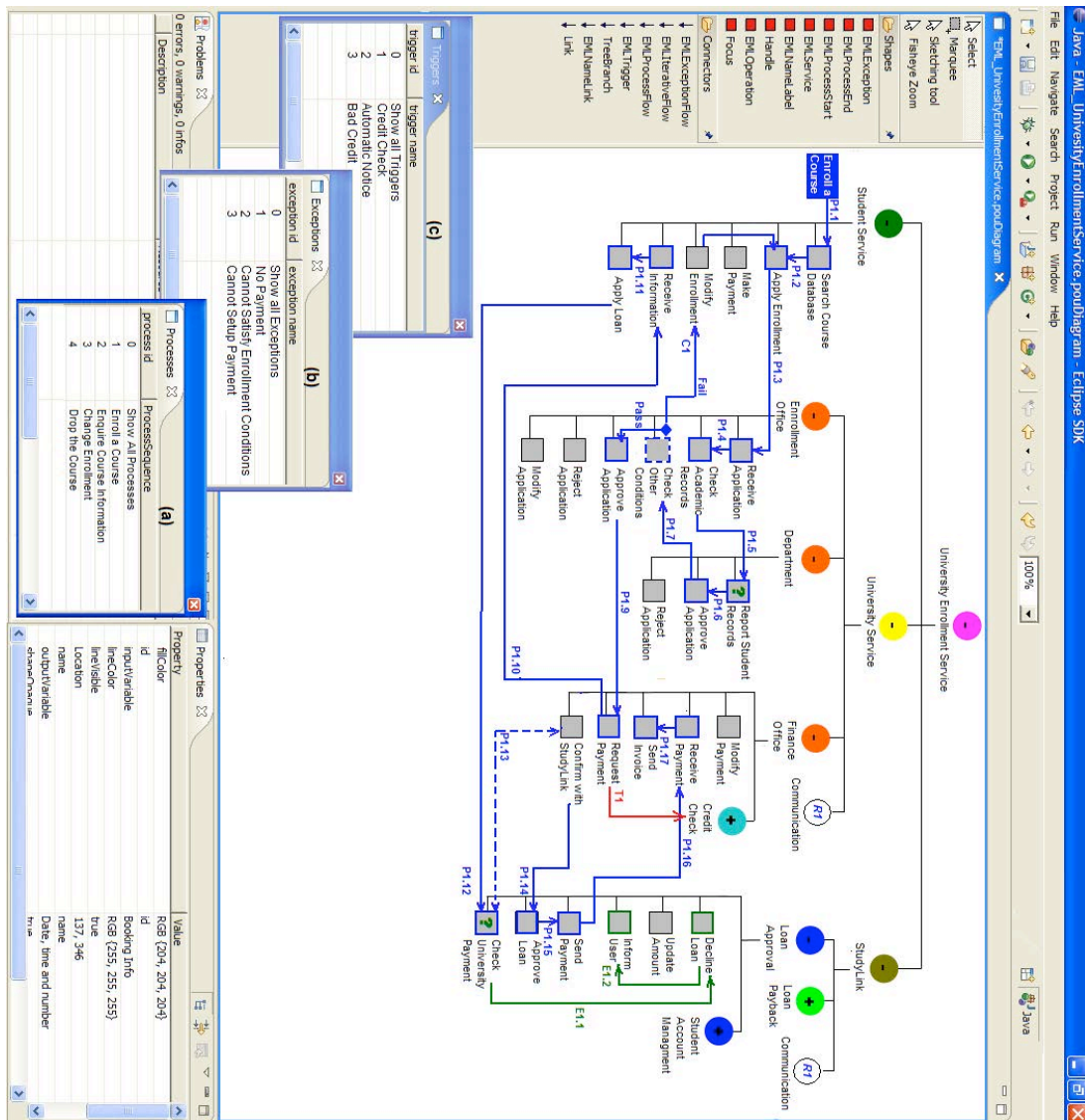
By modelling a business process as an overlay on the service tree, the designer is given a clear overview of both the system architecture and the process simultaneously. Processes can be elided as a way to mitigate the cobweb problem commonly seen in flow-based visual notations.

For example P1.1 to P1.17 in Figure 5.2 shows the *Enrol in a Course* process on the University Enrolment Service tree. The process starts with a process name followed by a process flow (blue arrow) representing the sequence. Each flow has a sequence number; for a complex process, users can use this to model concurrency /

synchronization. Involved operations or services have bold outline borders to help identify the track. Data is bound to a process flow to flow in or out of operations. In this process, the student uses *Search Course DB* to select the suitable course and *Applies Enrolment*. The enrolment officer *Receives Application* and *checks this student's Academic Records* with the *Department*. As soon as the *Department Reports the student's record* and *Approves the course Application*, the enrolment officer will *Check other Related Conditions* and ask the finance officer to *Request the Payment*. The student then *Applies Loan* and *StudyLink Checks University Payment* information with the Finance Office and decides if it *Approves or Declines the Loan*. If the university receives a payment from StudyLink, the finance officer confirms the enrolment and *Sends the Invoice* to the student.

EML supports specification of process iteration at different levels.

- (1) A single activity loop is represented as a dashed outline border. Attributes control the iteration (e.g. loop times, start and complete conditions, input/output data etc.). *Check Other Conditions* in Figure 5.2 is a single activity loop example. After the department *approves the course enrolment application* based on academic record, the enrolment office uses this function to repeat all the other related conditions (e.g. available seats in class, test time conflict, tutorial group assignment etc.).
- (2) A loop with two operations is represented using a dashed line with two arrowheads. Process P1.13 in Figure 5.2 shows iteration of the *Check University Payment* and *Confirm with StudyLink* operations. When StudyLink received the student loan application, they need to check all course related information with the university (e.g. student status, course fee amount, start and end date etc.). The process loops until a termination condition is met (all the information has been confirm).
- (3) If a loop involves more than three operations, a single arrowhead dashed line guides direction, linking different operations or services in a closed circuit.



**Figure 5.2: Using EML Overlays to Model the Enroll in a Course Process**

It is important to know if a specific event occurs or condition is met. Events and conditions are referred to as dependency relationships. In some cases, we can also treat internal (system) exceptions as triggers. An EML trigger layer can be used to solve dependency problems. T1 in Figure 5.2 shows how dependency information can be passed from one part of a process to another if a normal process flow is insufficient. The red single arrowhead trigger connector (T1) represents the dependency. In this example, when the Finance Office Requests the Payment from the student, they also need to Check student’s Credit. If the student has a scholarship, the requested payment amount may be changed. The user can define trigger conditions as attributes at each end of the connector to control the dependency. The start and end

point of a trigger can be a service, operation or process. Since EML uses a multi-layer structure, users can choose to combine their triggers with the process layers (as in this example) or separate them, using different views to reduce complexity.

The EML's exception overlay is used to model errors in transactions. A failure handling notation (question mark in the middle of an operation or service) specifies a transaction failure. Users can set up a start condition to discriminate different kinds of failures and activate appropriate exception handlers. An exception handling layer is constructed to model transaction error handling in detail. For example, Figure 5.2 shows the *Enrol a Course* process with two exception handlers overlaid. When the *Department* staff checks the student's academic record, an error handler is added to the operation (question mark in *Report Student Records*). If the student's previous academic record doesn't satisfy the course prerequisite, the application will be declined, which will drive the exception handler to carry out an alternative process (negotiate an alternative course with the student). A second exception handler is on the *Check University Payment* operation. If the student loan application cannot be fully confirmed by the *Finance Office*, the alternative is to *Decline Loan Application* and *Inform the Student*. Two green connectors (E1.1~E1.2) represent the exception flow.

The diamond shape in the above figure (Figure 5.2), attached to the boundary of *Check Other Condition*, is used to express a conditional flow. If the other course related conditions (e.g. an exam clash with another course) cannot be fully satisfied (Fail), the student will be informed to *Modify Enrolment*. Symbol C1 is an annotation used to describe such a conditional flow execution. Here, it may be a possible non-clashed exam time=table for the student to reference. If the student *Passes* the checking, the enrolment officer will then *Approve the Application*.

#### **5.4 BPMN Integration**

Due to the complexity of business processes, a single modelling notation is usually insufficient to satisfy all modelling needs. MaramaEML allows other business process modelling notations to be integrated to collaborate with EML to facilitate modelling of different structural and behavioural aspects. The integration of the BPMN notation



has been discussed in detail in Chapter 4 (MaramaEML, EML modelling tool implementation chapter).

For instance, in EML, data are bound to process flows via textual properties so as to reduce diagram complexity. However, sometimes a user may require this kind of information to be presented directly in the diagram. BPMN diagrams can represent well the internal flow sequence of data, but this kind of flow-based approach can easily cause diagram cobweb problems. An ideal solution is to provide the user with access to both diagram types. Our MaramaEML support tool includes linked BPMN and EML views.

Figure 5.3 shows a BPMN view for the “Enroll a Course” process. The *Student*, *Enrolment Office* and *Department* are described in three pools. Detailed process steps are:

- The Student “*Applies Enrolment*” by sending a message ([1] *I want to enrol this Course*) to Enrolment Office.
- The Enrolment Office “*Receives the Application*” and “*Check Academic Record*” with the Department by sending a message ([2] *Please Check this student’s Academic Record*).
- The Department staff “*Receive the Request*” and check student’s academic records. If the student passes the record checking, the staff “*Reports Back*” to the Enrolment Office by sending a message ([3] *This student can enrol*).
- The Enrolment Office “*Receives the Feedback*” and “*Approves the Enrolment internally*” by sending “*Approve Form memo*” to the Department.
- The Enrolment Office also requires the student to “*Check the Exam Impact Information*” by asking “[5] *does the exam time impact?*”

- The Student “*Receives the Request*” and “*Sends the Date Checking Request*” to Department asking exam date information (*[6] what’s the exam date for this course?*)
- The Department “*Receives the Date Checking Request*” and check the course related exam date and time information for the student.
- If there is no impact, the Department “*Sends internal Mail*” to the Enrolment Office asking (*[7] Prepare the Payment Invoice*) for the student.
- Then, the Department “*Send Feedback*” to the students, and telling the student (*[8] there is no exam time impact for this course, and I will inform the finance officer at Enrolment Office to prepare the payment invoice for you*).
- The Student “*Receives the Feedback*” and “*Sends a Message*” to the Enrolment Office saying (*[9] I need a payment invoice for this course*).
- The Enrolment Office “*Receives the Message*” and “*Send Invoice*” back to the student and saying (*[10] here is your invoice*).
- The Student “*Receives the Invoice*” and the make the payment. The enrolment process finish.

Figure 5.4 shows the multi-view collaboration between an EML view (a) and a BPMN view (b) to model the same enrolment process. From the EML view the user can obtain a clear service architecture and the process sequence, and from the BPMN view, he can also see the data transformation.

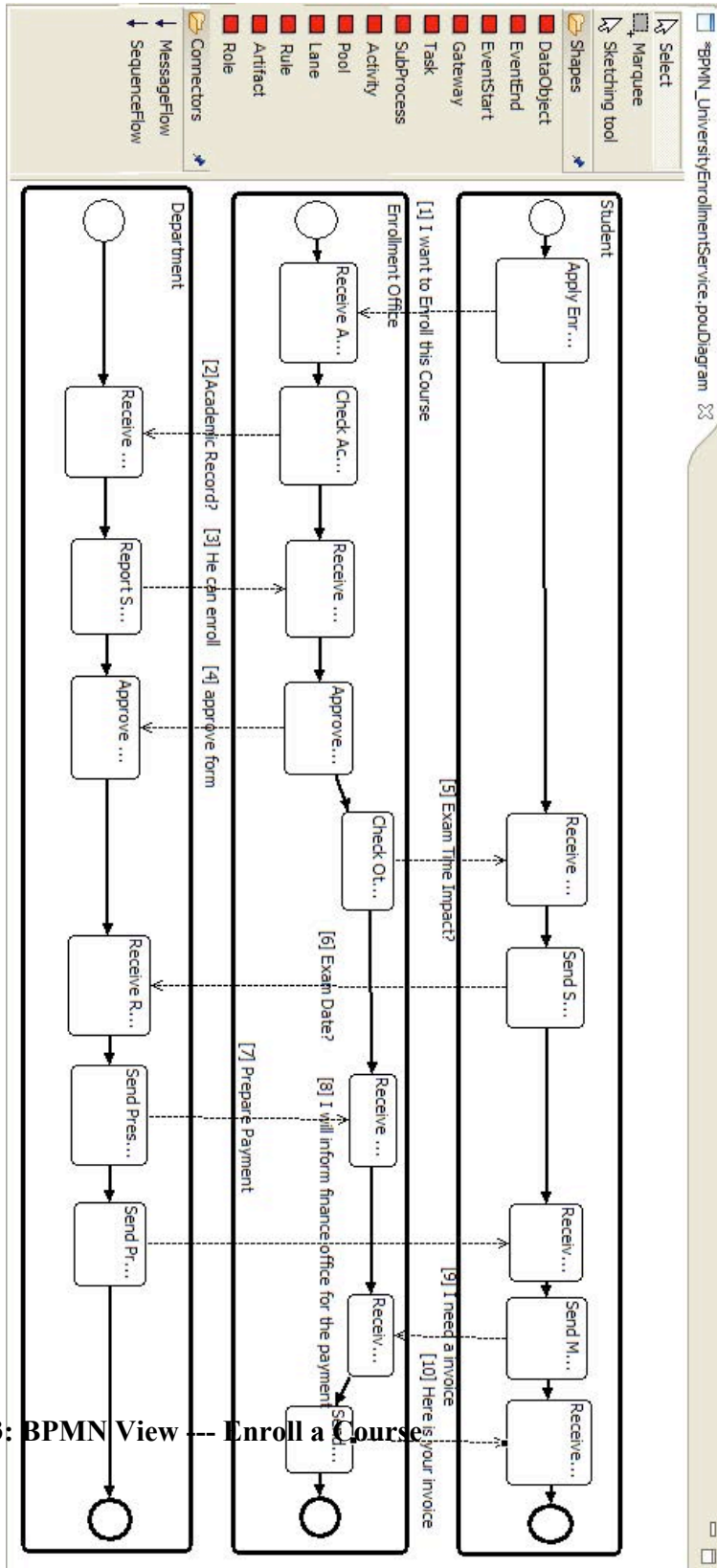


Figure 5.3: BPMN View -- Enroll a Course

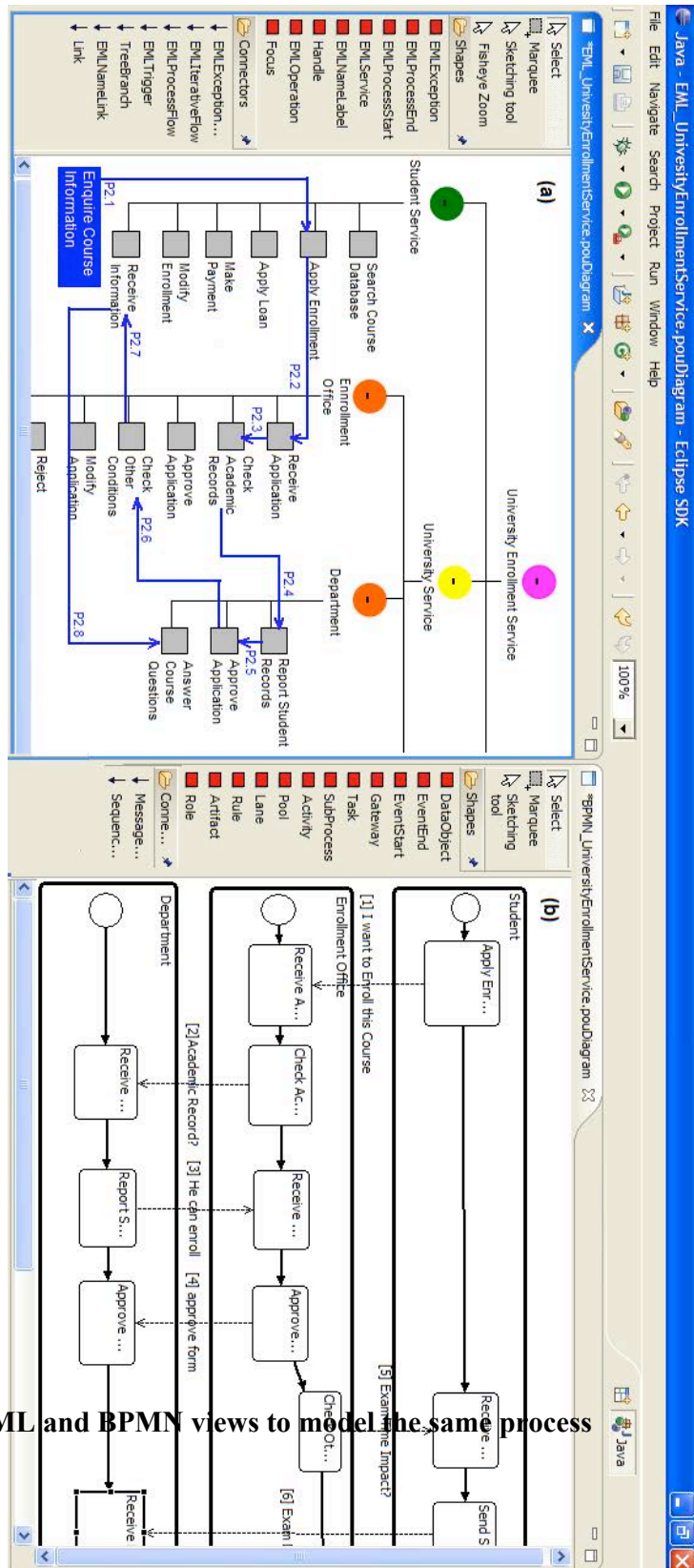


Figure 5.4: Using EML and BPMN views to model the same process

## 5.5 BPEL Generation and LTSA Validation

We use MaramaEML to generate Business Process Execution Language (BPEL) code and coordinate processes in a workflow engine. It allows us to export and integrate our EML structure with other BPEL compatible environments. To support code generation and process model validation we have developed a BPEL code generator and integrated an LTSA engine (Foster and Magee et al 2003) into MaramaEML to verify the correctness of EML models. As shown in Figure 5.5, the EML process layer (a) has been automatically compiled to executable BPEL code (b). Our code generator performs model dependency analysis and maps EML model constructs to structured BPEL activity constructs. The LTSA engine then verifies the correctness of the generated BPEL code. It compiles the BPEL code generated from EML specifications and displays the results in (d). If there is no compilation error, a LTS diagram (Labelled Transition System) is presented (c).

To generate the BPEL code, the user needs to:

1. Move the mouse to the EML tree structure area (a)
2. Right click the mouse to call the popup menu (e)
3. Select “Generate BPEL4WS from EML” function from the popup menu
  - The BPEL code will be automatically generated and displayed in area (b)

To verify the code, the user needs to:

1. Change view to “LTSA Perspective” from (h)
2. Open target BPEL code in area (b)
3. Select the Process Name in area (h)
4. Right click the process name in (h) to call the popup menu (g)
5. Select “Compile” function from (g)
  - The output from validation appears in (d)
  - The final LTS view appears in (g)

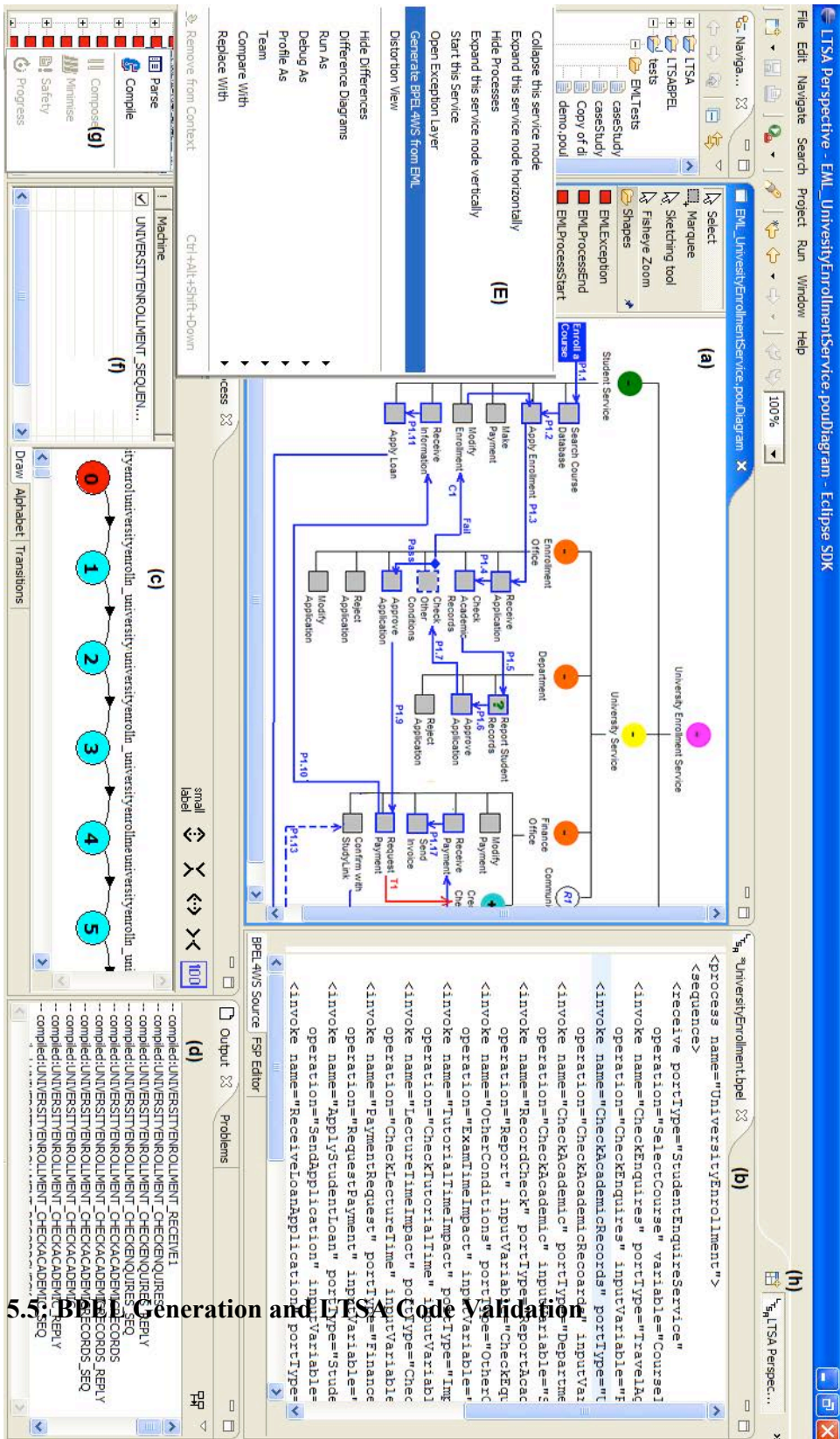


Figure 5.5: BPMN Generation and Java Code Validation

## 5.6 Zoomable and Fisheye Views

The EML's novel tree overlay structure has reduced the modeling complexity at a visual methodological level. However, due to the nature of enterprise complexity, sometimes the views can still be very large. At a technical level, in order to enhance EML's diagram navigability and understandability a zooming (radar view function) and a distortion-based fisheye zooming function would be helpful. We have developed those to add some complementary navigation support in EML.

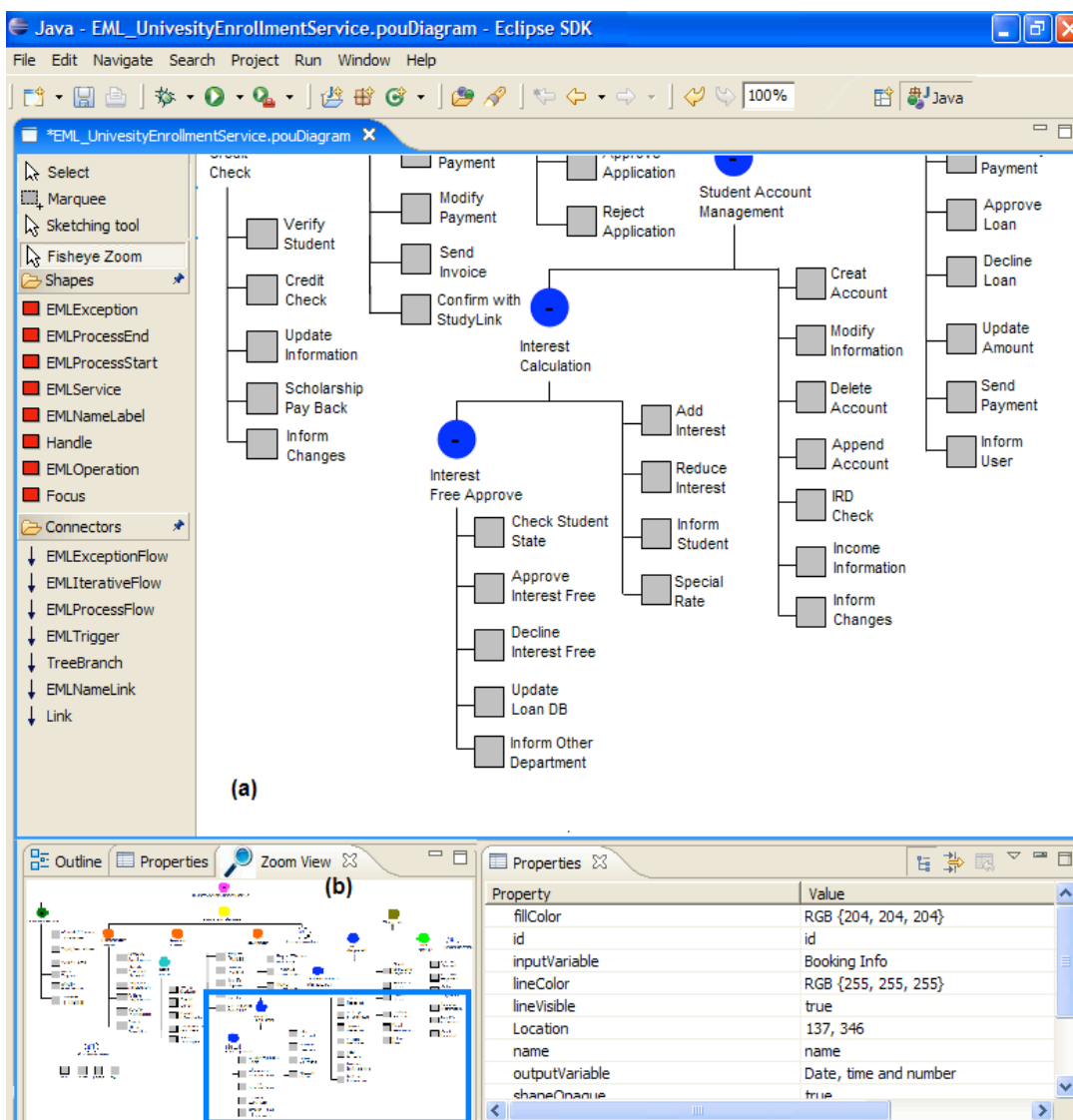
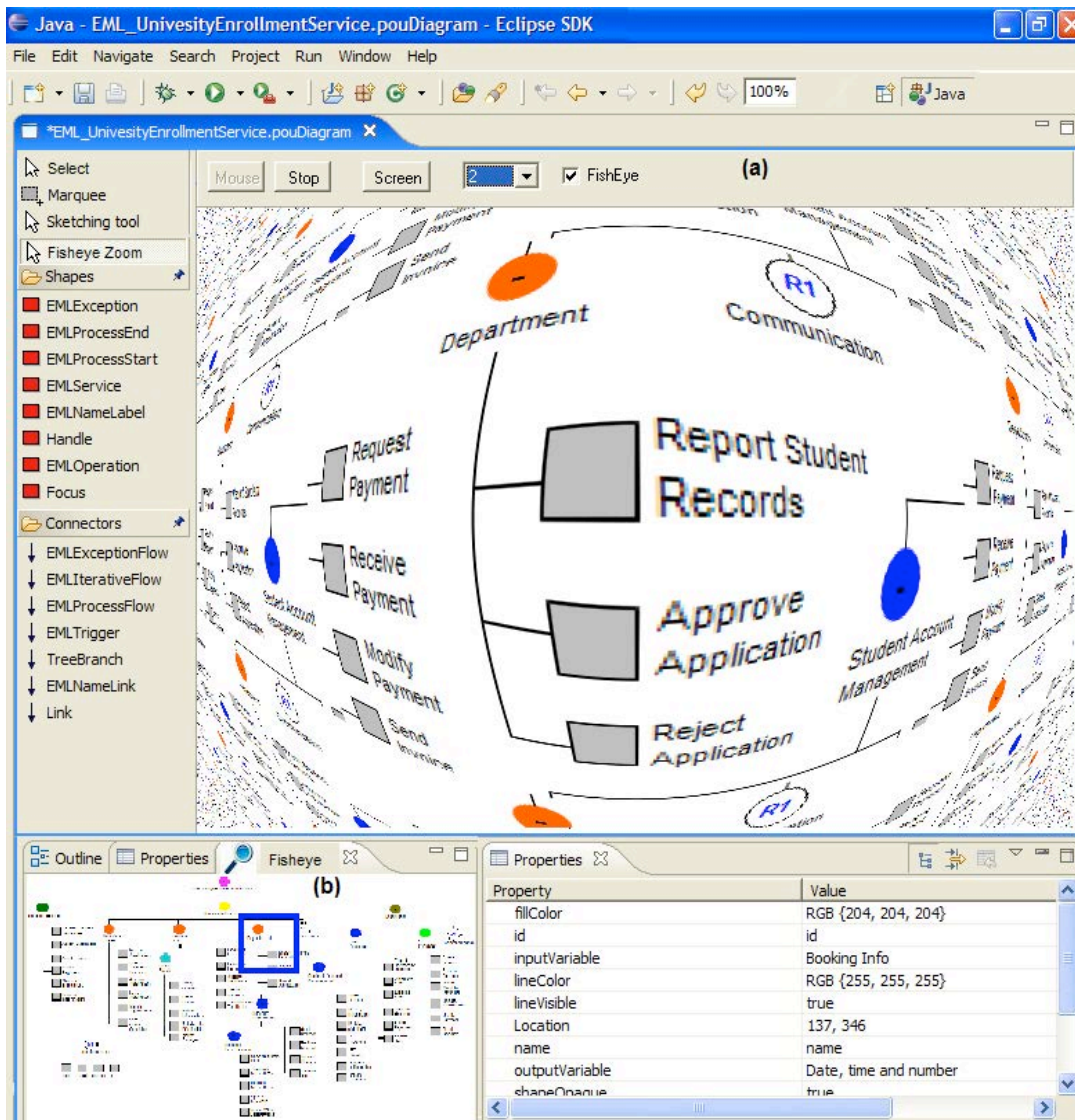


Figure 5.6 Zoomable View in MaramaEML

Figure 5.6 shows an EML zooming view (a). The user draws a “Radar square” (blue square) in the tree overview area (b). While the user moves the blue radar square, the components in area (a) (represented at the normal size) are moved to focus accordingly. By using this function, the user can have an overview of the whole tree structure, and in the mean time, be able to navigate to the detailed parts.



**Figure 5.7 Fisheye View in MaramaEML**

Figure 5.7 shows an EML fisheye view (a). The user draws a “fisheye area” (blue square) in area (b). Components in the blue square are represented in area (a) at a bigger size (*Department* Sub-tree), with the rest distorted with the degree of shrinkage increasing with the distance from the fisheye area. While the user moves the blue



radar square, the components in area (a) are moved accordingly to focus. In this example, the starting shrinkage degree is 2. At any stage, the user can change the value by selecting from the pull down menu.

By using the fisheye view, the user can have the freedom

- To show an area of interest quite large and with details
- To show other areas successively smaller and in less details
- To smoothly integrate local details and global context by repositioning and resizing elements.

## **5.7 Deployment**

There are several ways to deploy Business Process Execution Language (BPEL) definitions generated from EML visual process specifications. BPEL deployment engines are more commonly seen in Integrated Development Environments (IDEs) rather than in a standalone existence. Major IDEs include IBM's WebSphere Studio Application Developer, Microsoft's BizTalk server, and Eclipse's BPEL plug-in. BPEL processes can be assembled using IBM's WebSphere Studio Application Developer, and then deployed on WebSphere Integration Test Server. BPEL4WS specifications can be imported to the Microsoft's BizTalk server, and deployed to run in a production environment. BPEL processes can be copy-pasted and validated in an Eclipse BPEL project with automatically compiled WSDL interfaces and executed using its integrated Apache Orchestration Director Engine (ODE).

The above solutions introduce many overheads to deploy the BPEL processes generated by EML, including installation and coordination with different IDEs, with also compatible version requirements. After evaluating the feasibilities, we decided to use one of the two standalone BPEL execution engines: the Apache ODE and the IBM's PEWS4J, with the later deprecated as IBM has retired the project.

The Apache ODE implements the Web Services Business Process Execution Language (WS-BPEL) V2.0. Apache ODE executes BPEL processes, enabling their communications with other web services via passing of messages. Apache ODE

supports hot-deployment of BPEL processes. Deploying a business process in Apache ODE requires the BPEL files (\*.bpel, describing the process sequence, operation invocation and message passing), WSDL files (\*.wsdl, describing message types, port types, bindings and services for the process) and a deployment descriptor file (deploy.xml, an extra file that EML needs to generate so as to use Apache ODE to deploy the processes). These files need to be wrapped in an arbitrary folder and copy-pasted into the Apache ODE's processes deployment directory in Apache Tomcat (i.e. the TomcatInstalledDirectory/webapps/ode/WEB-INF/processes directory). The deployment starts automatically when Tomcat is running and a deployment file is generated, as shown in figure 5.8 and 5.9.

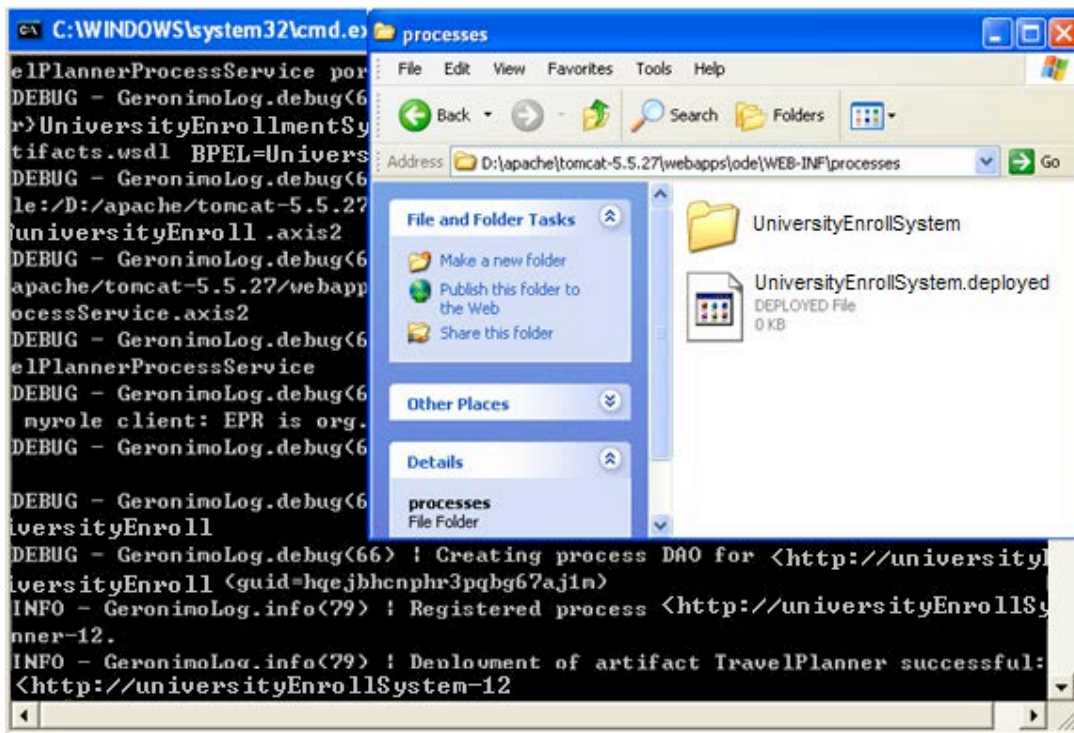
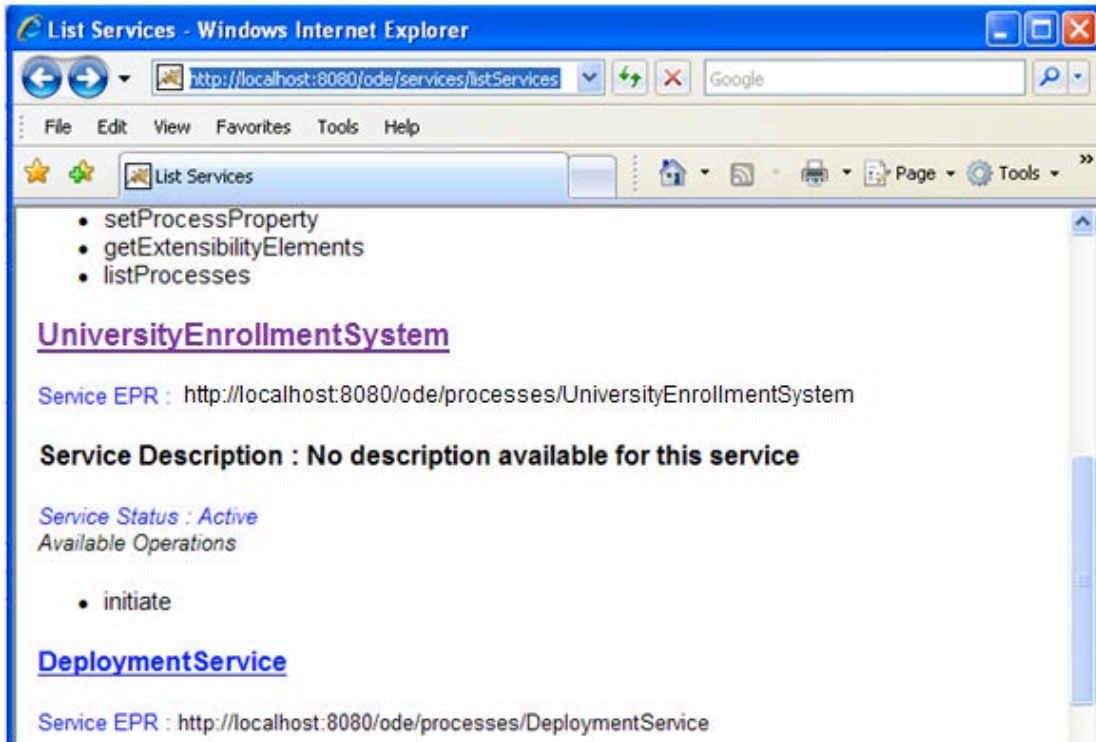


Figure 5.8 BPEL Deployment



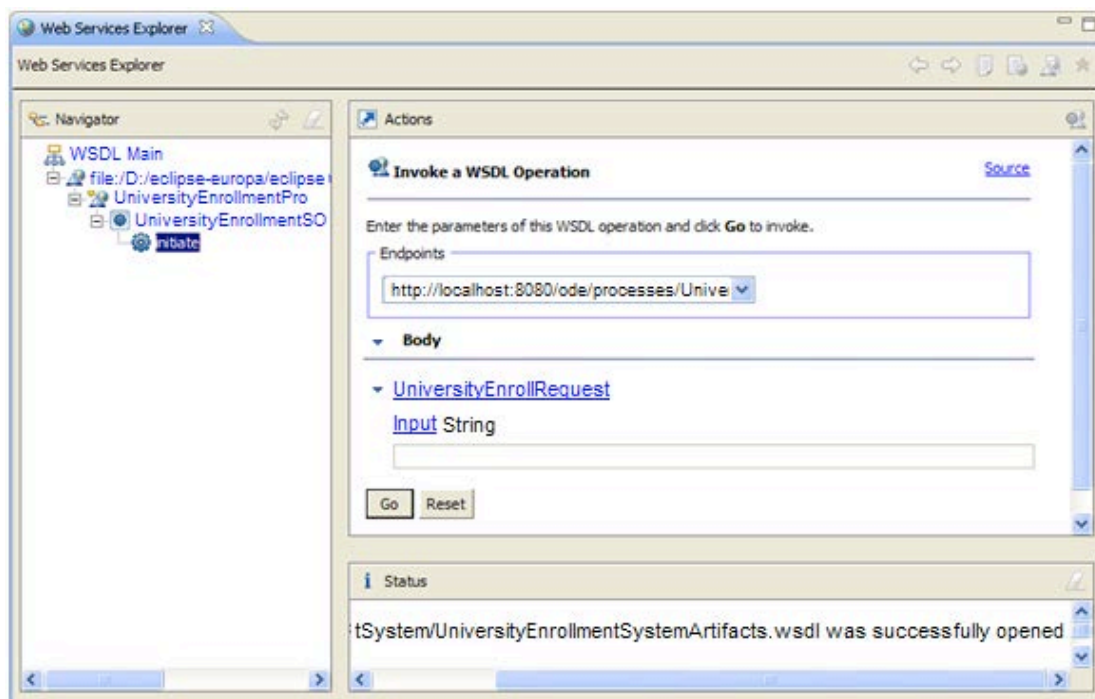
**Figure 5.9 Hot-deployment of a BPEL process in Apache ODE**

An example of EML generated process specification contains the following files which are needed for Apache ODE deployment:

1. BPEL
2. WSDL
3. Deployment descriptor file

Before a BPEL process can invoke a Partner (other service), it needs to first define a PartnerLink to reference the Partner's WSDL interface. Furthermore, each BPEL process needs to define a PartnerLink that represents itself which points to its own WSDL interface. For every BPEL process, it must have at least one PartnerLink that describes itself. If the process invokes other services, each of those services would require a PartnerLink definition. These can be inferred from EML's service trees and process overlays, so no explicit modelling of partnerlinks are required.

A deployed process is provided with a SOAP interface and a WSDL file, and thus can be invoked by a requesting web service client. We can use the Eclipse Web Services Explorer tool (as shown in 5.10) to test the deployed process. This Eclipse plug-in can be started by right-clicking the process WSDL file and selecting to open Web Services → Test with Web Services Explorer. Using the Web Services Explorer interface, the user can select a process operation and enter some input message to invoke the operation. Web Service clients can also be generated using the Eclipse's Web Services Generate Client tool in the Eclipse environment to communicate with the business processes.



**Figure 5.10 Testing process WSDL interface using the Eclipse Web Services Explorer**

## 5.8 Summary

We have described a University Enrolment System example in this chapter. We have used our evolving Pounamu and Marama meta-tools to develop the EML modelling environment. We specified the EML domain-specific visual language notation and meta-model and generated Eclipse-based editors from these to realise the basic support environment. The tree layout, overlays and distortion-based displays are all

implemented as complex visual event handlers. The integration of EML with the BPMN notation, the code generation of BPEL, and the integration of the LSTA engine were implemented through event-driven, meta-model and model level data queries and updates.

# Chapter 6

## EVALUATION

---

Evaluation has played a very important role in the entire EML and MaramaEML design and implementation process. Three different kinds of evaluation methods have been applied to the visual modeling language and its support tool as we progressed on the research project. We continuously used the results from these evaluations to refine the functionality, usability and other related quality attributes of the language and environment. This chapter describes the evaluation approach taken for EML and MaramaEML. An overview of the evaluation approaches is introduced in section 6.1. The first evaluation, using a cognitive dimensions walkthrough, is described in section 6.2. In section 6.3, we describe a second informal evaluation with a small number of experienced domain experts (the target users have Software Engineering and/or Computer Science backgrounds). A large formal end user evaluation and a statistical analysis of its results are reported in section 6.4.

Our formal end user evaluation (titled “A Visual Language and Support Tool for Business Process Modeling”) was approved by University of Auckland Human Participants Ethics Committee (reference number 2007/178).

### 6.1 Evaluation Mechanisms Overview

Rather than simply evaluating MaramaEML once the design was completed the approach we have taken has been to use a variety of evaluation approaches during the design and implementation and make improvements to the language and toolset by analyzing the evaluation outcomes. To do this we conducted three different evaluations for EML and MaramaEML. These evaluations spanned from the early language design stage to the ultimate software tool release phase. The targeted users included EML designers, computer science and software engineering domain experts and business end users.

The first round of evaluation was an extensive cognitive dimensions (CD) analysis (Green and Peter 1996) which was used to guide the early design and implementation of EML. It was undertaken by the EML designer with closeness of mapping and hidden dependency mitigation emphasised. A list of related CD design guidelines and trade-off principles were applied to an early version of EML. The visual language was improved as a result of analysing the CD outcomes.

The second evaluation was a small scale task-based end-user evaluation of an early released version. A group of Computer Science and Software Engineering tool design experts were selected to perform this evaluation and provide their professional feedback. The objective was to professionally assess how easy it was to learn to use MaramaEML and how efficiently it could solve the diagram complexity problem. We refined our software tool and modelling language based on the feedback from this evaluation.

The third evaluation was a large formal end user evaluation of the most recent release. More than thirty end users with different backgrounds were selected for this evaluation. The results suggest that MaramaEML is very straightforward to use and understand. Users feel the tree overlay method greatly reduces the complexity of modelling business processes compared to using only the conventional BPMN views. The automatic code generation and multi-view collaboration mechanisms were seen as enhancing the modelling strength. The zooming and fisheye function was seen as being very easy to use and increasing the tool's navigation ability. We made a final round of visual language and software tool improvements guided by the feedback obtained.

## **6.2 Cognitive Dimensions for Early Validation and Design Assistance**

Applying Psychological principles to Computer Science has been an active area of research since the 1970s (Green and Peter 1996). Psychology of Programming, as a field, was established in 1987 to coordinate research in the areas of cognitive psychology in software development.

The Cognitive Dimensions (CDs) of Notations approach (Green and Peter 1996) is a popular, psychologically based, heuristic framework initially created by Thomas Green and Marian Petre that is designed for quickly and easily evaluating a visual language environment. It sets out a small vocabulary of terms designed to capture the cognitively relevant aspects of structures, and shows how they can be traded off against each other. They are not intended to provide a rigorous analysis, but instead give the designer a rough idea of some of the human factor issues inherent in the system. They allow the designer to get a general feel for the characteristics of the system before or instead of running expensive usability tests. From there, the designer can come up with potential tests and changes that will improve the system (Dillon 2001; Hartson and Andre et al 2003; Recker and Rosemann et al 2009). Microsoft has applied cognitive dimensions, to their C# and .NET development tools (Chappell 2007).

We use the CD approach to help design our Enterprise Modelling Language. The cognitive dimensions take a complete view of a visual environment, covering both the visual notation and its environmental support. The notation is the textual or graphical view into the programming structure. The environment is the way the notation is manipulated by end users. The system is defined as both the notation and the environment. There are 13 dimensions, each representing an aspect of the system which has an impact on the ability of users to work with it. Not all of them are of strong relevance to EML and its integrated support environment. Ten most relevant dimensions were chosen for our analysis task and the results of investigating them against our MaramaEML support environment are described below. For each of the dimensions, we discuss its definition and then its relevance to EML.

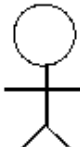
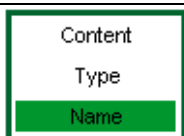
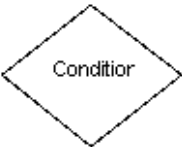






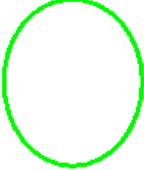





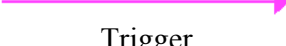
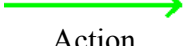
### **6.2.1 Consistency**

*Consistency* (similar semantics are expressed in similar syntactic forms) refers to the “guessability” of a system. Given knowledge of some of the system structure, how much of the rest can be guessed? The simplicity brought about by consistency is because there are not that many types of definitions, expressions, etc. to learn.



- **Before CD Evaluation**

Figure 6.1 lists the most commonly used notational elements in an early version of EML. As it shows, this version of EML notations comprised a set of dis-similar semantic elements. For example, data flow, control flow, action flow and trigger all use different connection shapes and colours. The process execution conditions (successful finish, failed and aborted) are represented in different shapes outside the service or task icon.

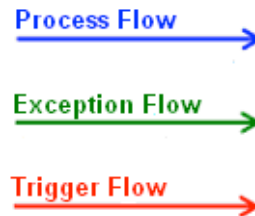
 Actor Actor	 Data Set	 Condition	 Successful Finish	 Failed	 Aborted
 Service	 Task	 Constraint	 Action Focus	 Assign	 Delay
 ForEach	 Data Flow	 Control Flow	 Trigger	 Action Transition	

**Figure 6.1 Selected EML Notation Examples (before CD)**

- **After CD Evaluation**

To enhance consistency, based on the CD results, we changed all the process, trigger and exception flows into the same shape and used different colours to separate them. Figure 6.2 represents the improvements of the flow notations after the CD analysis. The process execution conditions have been integrated into the service or task node. In the improved version of the EML notations, the data and control flows have been combined into the process flow (blue line with single arrowhead). The exception flow (original Action Transition) is a green line with a single arrowhead. The trigger flow is a red line with a single arrowhead. Please refer to chapter 3 for a complete list of the new EML

notations. The new EML underlying structural tree provides a consistent framework on which similar semantic operations (standard process flow, triggers and exceptions) are overlaid using similar syntactic forms (flow, distinguished by colour).



**Figure 6.2 Improved Flow Notations (after CD)**

### **6.2.2 Visibility & Juxtaposability**

*Visibility & Juxtaposability* (ability to view components easily): A system with low visibility makes it cognitively difficult to bring related structures into view. Juxtaposability refers to the ability to view objects side-by-side. These dimensions primarily focus on an editing environment.

- **Before CD Evaluation**

The early version of EML was based on pen and paper. It didn't have a software tool to support it. The visibility and juxtaposability (based on paper) were both very poor. It didn't have multi-view support functions with the language.

- **After CD Evaluation**

In order to enhance the visibility and juxtaposability, a software tool to support EML modelling was required. Thus, we developed an Eclipse based EML support tool - MaramaEML. In MaramaEML different modelling notation views can be juxtaposed. The zooming and fisheye viewer supports a high degree of visibility within EML views, even for very large diagrams. Please see chapter 4 (software tool implementation chapter) for more detailed description of the support environment.

### 6.2.3 Premature Commitment

*Premature commitment* (constraint on the order of doing things): When drafting a thesis structure in a paper with a pen, you need to make sure to leave enough room for the detailed bullet points between the headings. This is an example of premature commitment, where the user is required to make decisions when not ready.

- **Before CD Evaluation**

The user could only use EML to model a system by pen and paper. We didn't have software to support the multi-notational modelling function (e.g. EML & BPMN). Even for EML itself, there was not much freedom to transfer between views, processes and different layers based on paper. As soon as the structure was drafted, it was very hard to make any changes.

- **After CD Evaluation**

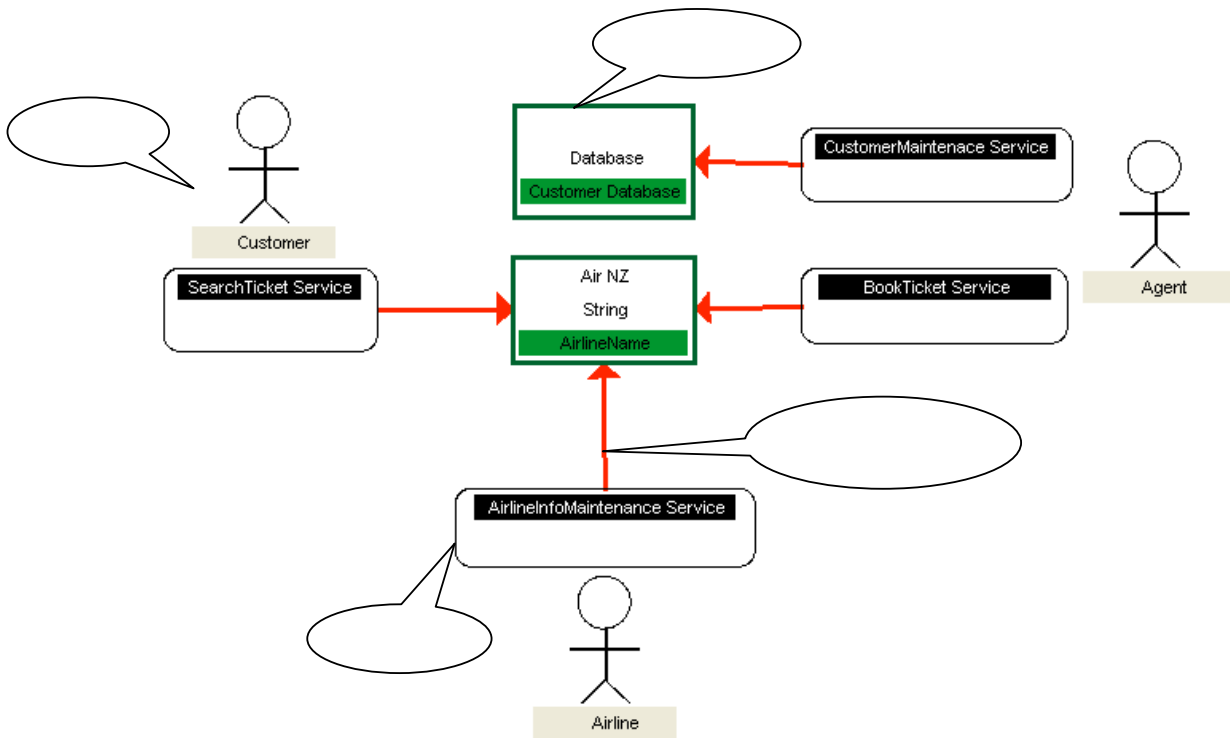
The user has considerable freedom to model a business process using any EML, BPMN and Form-Chart notation via the EML's new software tool MaramaEML. In the EML view, the user can freely traverse through the tree structure view and the business process, triggers and transaction layer, or even integrate them together. Chapter 5 has detailed case study to demonstrate the above features. However, the user needs to define the business tree structure first and then construct process overlays.

### 6.2.4 Hidden Dependencies

*Hidden dependencies* (important links between entities are not visible): Hidden dependencies are relationships between objects in the system that are implicit and difficult to uncover. The cognitive dimensions are meant to be evaluated in terms of both the notation and the environment used to manipulate that notation.

- **Before CD Evaluation**

The early version of EML had strong hidden dependencies. Figure 6.3 represents a service view diagram example in the early version of EML. It mainly modeled the communication between the database and the service. In this example, all of the customer, agent and airline services try to connect with the airline database to gather the information, and the agent also tries to check the customer database to verify the user's credit information. As the example shows, all the communications among the three parties were automatically hidden, and the database structure and their information inside were also invisible to the user.



**Figure 6.3 Service View Diagram in Early Version of EML**

- **After CD Evaluation**

By default some dependencies are still hidden, e.g. data bound to the process flows, and trigger and exception flows are normally not shown in a process layer. However, most information is readily accessible via property sheets in MaramaEML. The propriety window and the multi-language modelling support can represent these kinds

of abstractions with MaramaEML’s tool support. It is possible to show all the dependencies in the same layer if required.

Property	Value
*Model Elements	
Actors	[Travel Agent]
ChildService	[Hotel Booking; Air Ticket Booking]
Documentation	[Customer Database; Booking Database]
ElisionType	Extend
ExtendedProperties	[None]
fillColor	RGB {192, 192, 192}
Id	Service 1
Input	[Client ID]
lineColor	RGB {0, 0, 0}
Location	207, 61
LoopType	More than Two Services Loop
Name	TravelPlanner
Operations	[Client Credit Check; Available Room Check; Available Airline Search]
Output	[Booking Request ]
ParentService	[Enterprise Service]
ReuseId	N/A
ReuseStatus	False
ServiceType	Service
Sign	Yes
Size	40, 40
Status	Other

The Status dropdown menu is open, showing the following options: Aborted, Other (selected), Failed, Skipped, Finished.

**Figure 6.4: Property Sheet Example for Travel Planner Service**

Figure 6.4 shows a screen dump of the Travel Planner service’s property sheet in MaramaEML. It covers a list of dependencies for this service node. In this example, the *actor* of this service is “Travel Agent”. The travel planner service has two *child services* (“Hotel Booking” and “Air Ticket Booking”). The *documentations* related to this node are “Customer Database” and “Booking Database”. The *elision type* is “Extend”, and there is “None” *expended properties* for this service. The *fillColour* for the service notation is “RGB{192,192,192}”, and *lineColour* for the notation boundary is “RGB{0,0,0}”. The *location* (on the screen) of this service node is “207, 61”.

In this booking process, the *input* of the service is “Client ID” and the *ID* for this node is “Service 1”. The *name* of the service is “Travel Planner”. “More than Two Service

Loop” is shown in the *loop type*. There are three *operations* in this service (“Client Credit Check”, “Available Room Check” and “Available Airline Search”). The service sends “Booking Request” to the other services or operations as the *output*. “Enterprise Service” is the *parent service* of travel planner. This service node has not been reused. So the *reuse status* is “False” and the *reuse ID* is “N/A”. The *type* is “Service” and it has the *sign*. The *size* of the shape is “40, 40”. The *status* of this service is “Other”.

### 6.2.5 Error Proneness

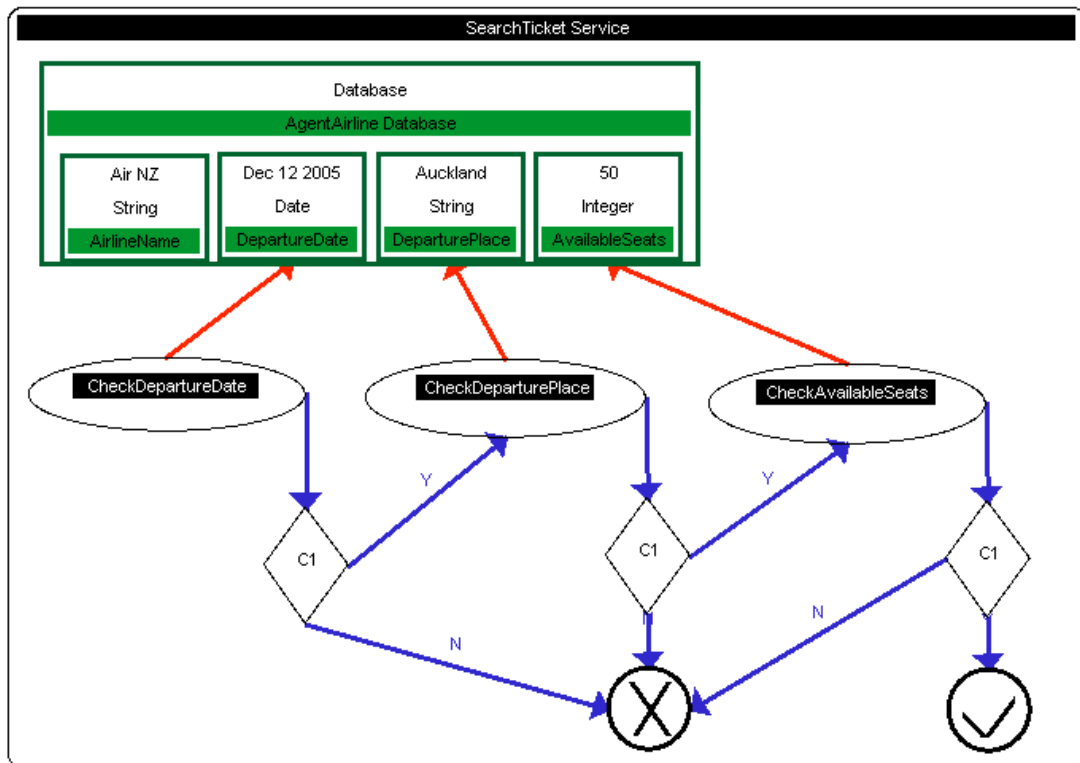
*Error proneness* (the notation invites mistakes and the system gives little protection)

- **Before CD Evaluation**

The early version of EML had several areas of error proneness. Figure 6.5 represents a service integration view example in the early version of EML. It describes the booking air ticket process. Three web service nodes (CheckDepartureDate Service, CheckDeparturePlace Service and CheckAvailableSeat Service) talk to the Airline Database and then make the final decision (book the ticket or cancel the booking). In this example, there was no connector constraint to limit the connections. Basically, the user can use any connector to link any notations.

- **After CD Evaluation**

In order to reduce error proneness, the new software tool MaramaEML enforces connectivity constraints and provides design feedback to users on the correctness of notational usage. A list of extra building constraints have been added into the EML design rules. The EML notation has become simpler and better defined with high level business modelling graphical representations.



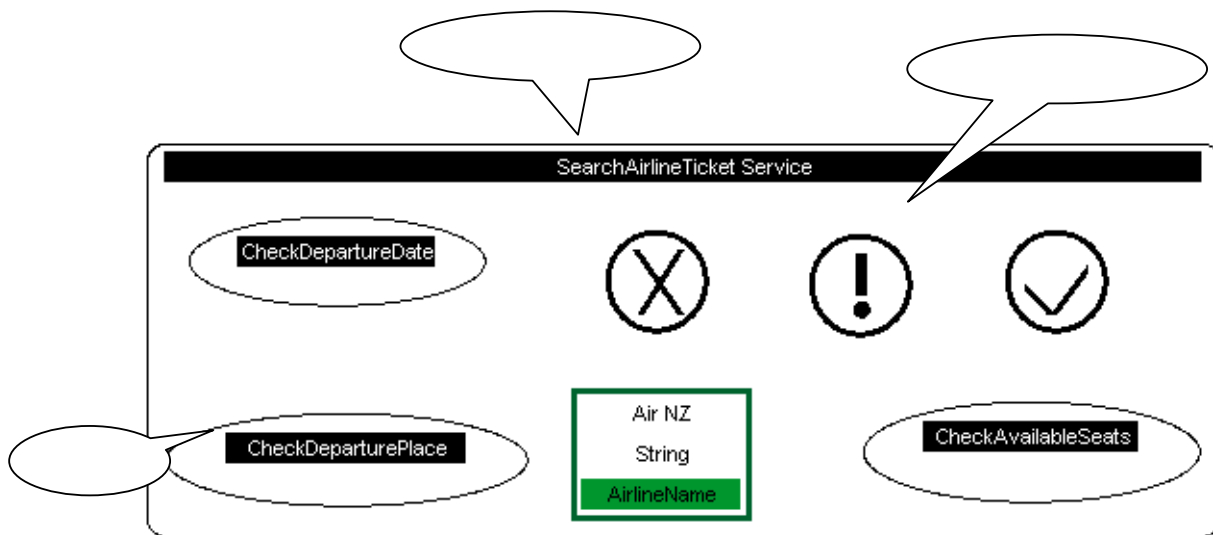
**Figure 6.5: Service Integration View in Early Version of EML**

### 6.2.6 Abstraction

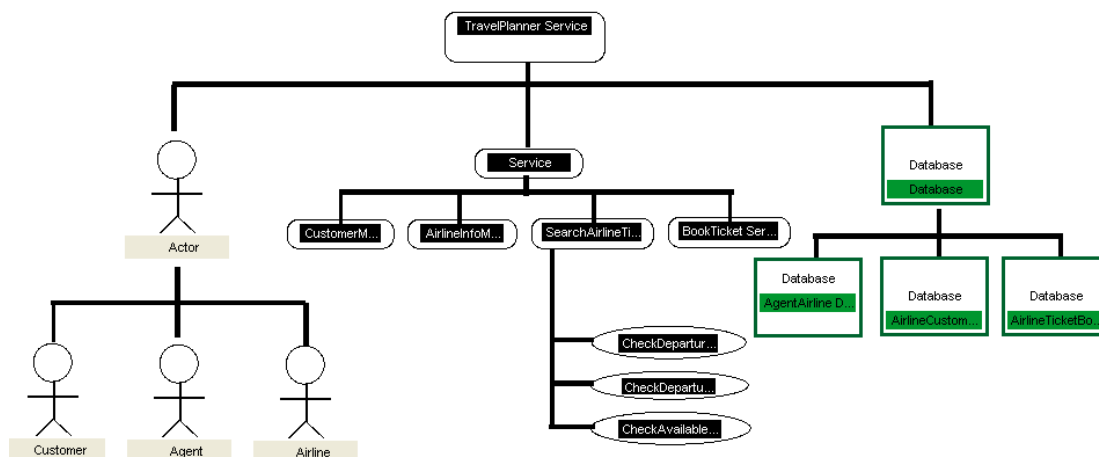
*Abstraction* (types and availability of abstraction mechanisms): The abstraction gradient represents whether users are required to learn abstractions before effectively using the system and whether they are allowed to use abstractions if they want to.

- **Before CD Evaluation**

Early versions of EML used a mixed form and tree overlay based metaphors. The user had to use tree layout to represent the overall structure (as shown in Figure 6.7) and use form layout (as shown in Figure 6.6) to represent the detailed information inside each service and task node. By doing this, it provided both visual metaphors to the user at the same time. But it also required a high abstraction gradient due to the need to learn both notations and relate them together.



**Figure 6.6: Form Based Service View in Early version of EML**



**Figure 6.7 Tree Layout for the overall Structure**

- **After CD Evaluation**

In order to enhance the abstraction ability, we decide to use the tree layout in all modelling situations. The solid tree layout is very easy for the user to understand providing minimal abstraction gradient for the target end-users. Please refer to Chapter 3 (Enterprise Modelling Language Chapter) for detailed layout description in EML. The refined EML is a high level process modelling language but the metaphors it uses are very business-oriented and tailored for the enterprise domain.



### 6.2.7 Secondary Notation

*Secondary notation* (extra information in means other than formal syntax): In EML, we integrate shapes, colours, text descriptions and tree layout structure together to convey information.

- **Before CD Evaluation**

By using pen and paper, the user can easily extend the notation in the diagram to convey extra information.

- **After CD Evaluation**

In MaramaEML, sketch annotations can also be added to diagrams to convey extra information.

### 6.2.8 Closeness of mapping

*Closeness of mapping* (closeness of representation to domain): EML uses a tree metaphor to represent the service construction. This hierarchical structure is a natural way to model business structure and users are familiar with using it to model complex organizational hierarchies. Business processes are constructed using a flow-based overlay metaphor on top of the tree structure providing good closeness of mapping to process sequencing.

- **Before CD Evaluation**

The early version of EML tree did not have any elision function. The tree overlay (e.g. in Figure 6.7) had to represent all the nodes in one screen. When the system became complicated, it was very hard for the user to understand the whole structure, as the notational approach was not scalable.

- **After CD Evaluation**

In order to solve this problem, an elision function has been added in the new EML system (please see Section 3.2.4 in Chapter 3 for detailed notation elision description). A minus (-) symbol indicates that all activities in the

service have been expanded. A plus (+) symbol indicates that part or all of the sub-tasks (services and operations) are elided. The elision techniques allow users to focus on one process at a time, minimising their cognitive load and maximising the ability to relate the diagram to the actual business process being examined.

### 6.2.9 Diffuseness

*Diffuseness* (verbosity of language): How many symbols or how much space does the notation require to produce a certain result or express a meaning?

- **Before CD Evaluation**

The early version of EML used several different notations, including the tree and form metaphors (as showed Figure 6.6 and 6.7), and a large number of notational elements (as shown in Figure 6.1). The language was thus verbose, and added extra leaning efforts for the users.

- **After CD Evaluation**

The refined EML evidently reduced the number of notational elements to a large extent and eliminated the form based metaphor. The tree overlay becomes the only metaphor in the language. The new EML uses a terse set of language elements and hence it is easy to learn.

### 6.2.10 Hard mental operations

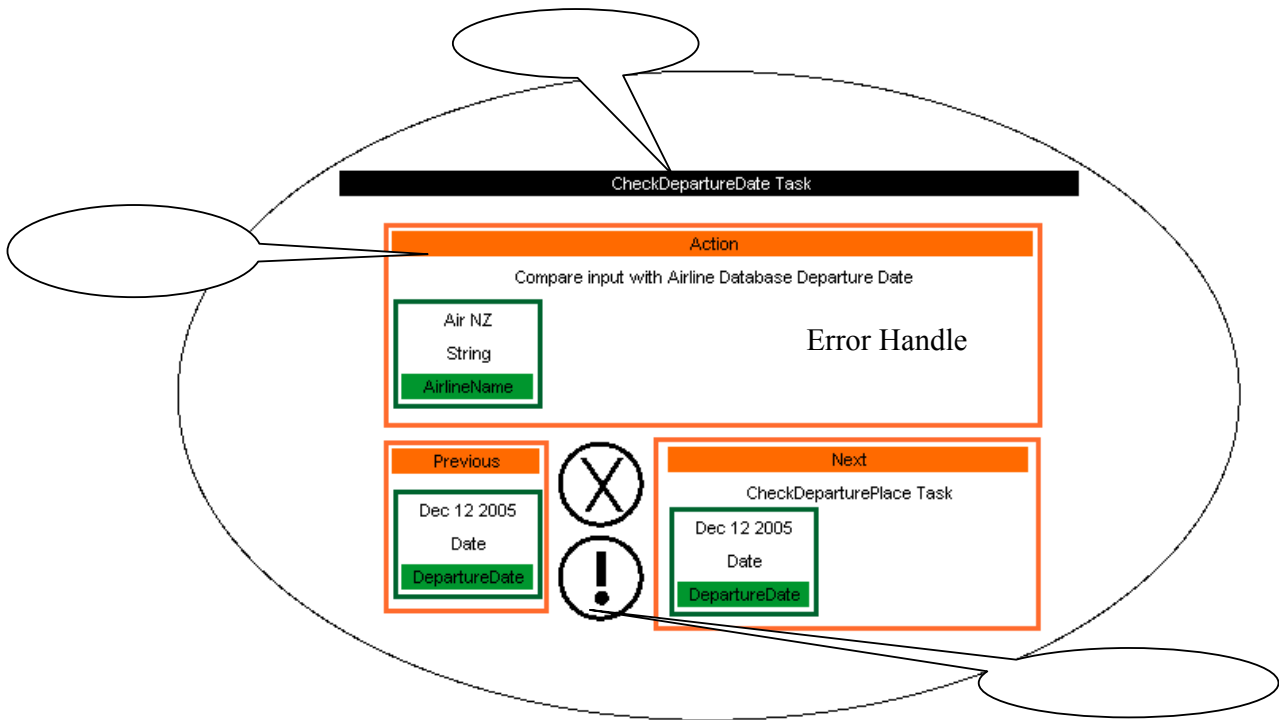
*Hard mental operations* (high demand on cognitive resources): How much hard mental processing lies at the notational level, rather than at the semantic level? Are there places where the user needs to resort to fingers or penciled annotation to keep track of what's happening?

- **Before CD Evaluation**

Figure 6.8 shows an exception handler example in the early version of EML. The constraint and service structure are represented using the same form based layout. The exception handler was described in the same manner as the

ts

execution state. This was very confusing for the end user, requiring hard mental operations to distinguish them.



**Figure 6.8: Exception View in Early version of EML**

- **After CD Evaluation**

In the refined EML, different states and components of a business process are well discriminated through the use of the tree-based hierarchy, process overlay, dependency trigger and different layers of exception handlers. The complexity of a business process has been successfully reduced in the EML multi-layer structure through its elision and overly techniques.

### 6.3 Early Evaluation with Experienced Tool Developers

One important result from the CD analysis was the demand for a software support tool for EML. After we completed the first version of MaramaEML, we selected a group of Computer Science and Software Engineering experts to carry out a task-based end-user evaluation of the refined EML and an early version MaramaEML prototype. The objective was to assess how easy it is to learn to use EML and its support tool and

(a)

how efficiently it can solve the diagram complexity problem. This was an informal evaluation and the result was used to refine the MaramaEML design.

### 6.3.1 Evaluation Environment

(c)

We used the refined version of EML (2.0) and an early prototype of the software tool MaramaEML (version 1.0) to perform this round of evaluation. The main focus was on the functionality. Although it was fully featured for most of the functions, the shape definitions were quite simple. However, the results were promising. The main functions of this version of MaramaEML included:

- EML tree and process overlay modelling ability.

Figure 6.9 shows an E-mail Voting process overlay (green) example on an EML tree (black tree). The system provides an individual modelling function for the EML tree layout, process and trigger overlays in area (a). The EML shape and connector tools are listed in area (b) and (c). The user can directly drag and drop a selected component into the EML diagram. All business process overlays are recorded as a textual list in a supporting window (d).

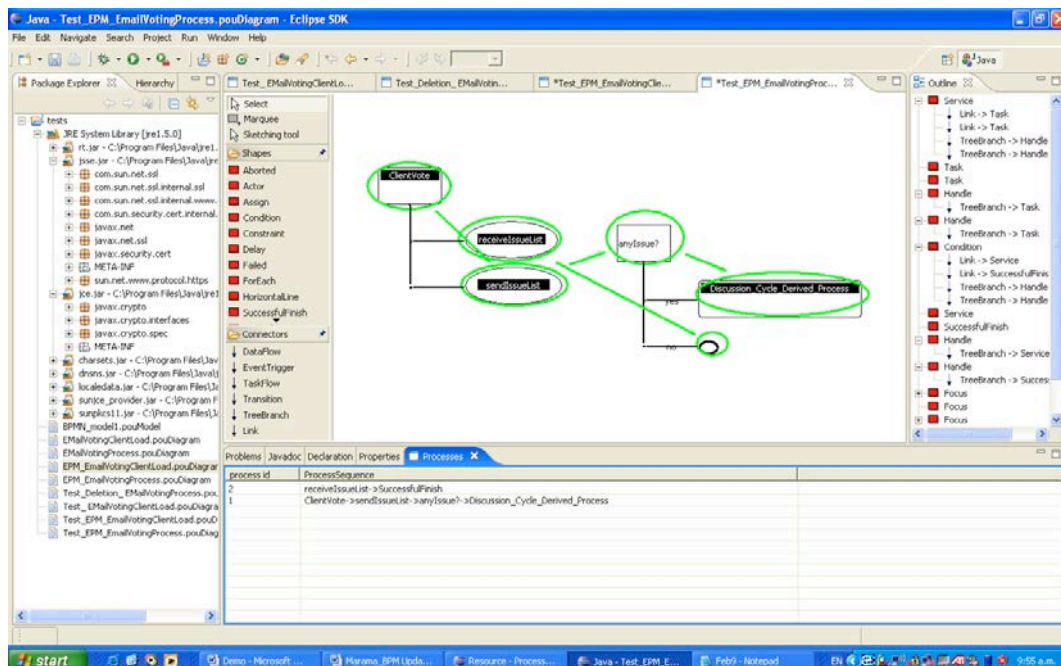


Figure 6.9: A process Overlay on EML Tree in MaramaEML 1.0

(a)

(b)

- Basic BPMN modelling ability.

MaramaEML 1.0 also provided individual modelling support for BPMN.

- (c) Figure 6.10 shows an E-mail Voting example in a BPMN diagram. As we mentioned, this version of software mainly focussed on functionalities instead of a “polished” user interface design. The shapes provided in the BPMN view were thus limited, but the tool covers most commonly used BPMN components e.g. Pool, Task, Process. In this example, the BPMN diagram is modelled in (a); its shapes and connectors are in (b) and (c). The properties of each component are managed in (d).

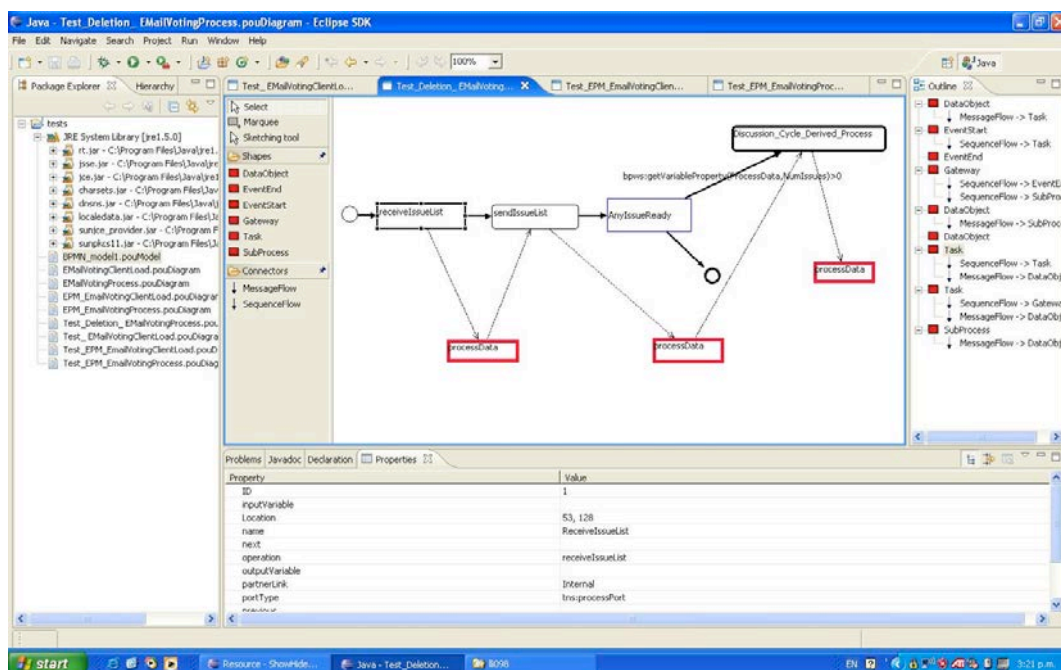
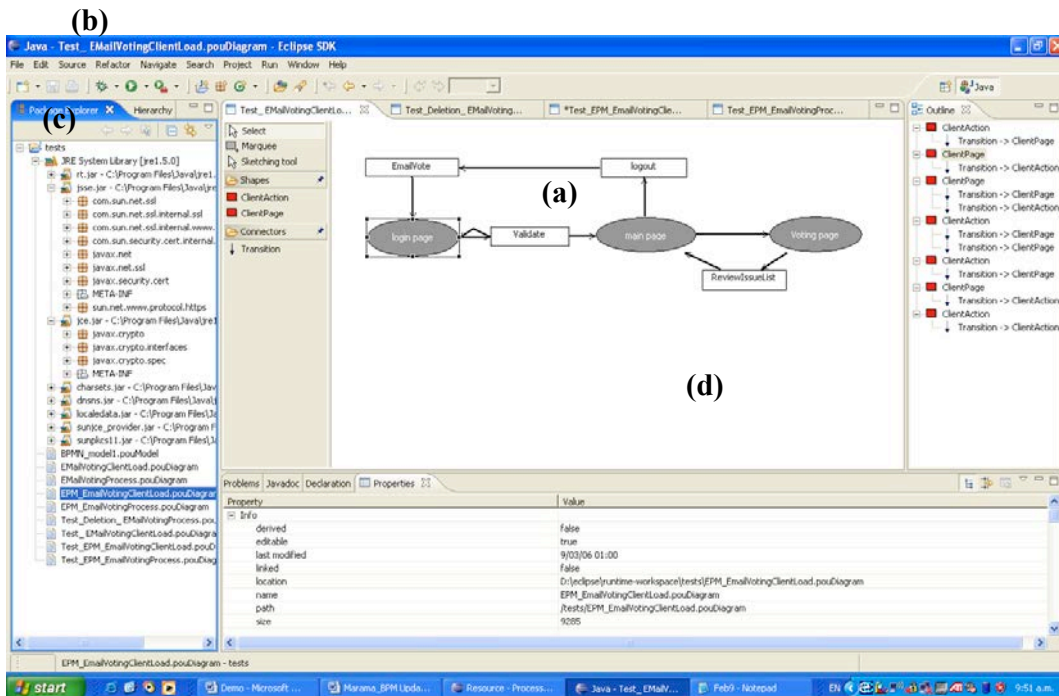


Figure 6.10: A BPMN Diagram in MaramaEML 1.0

- Form Chart Modelling ability.

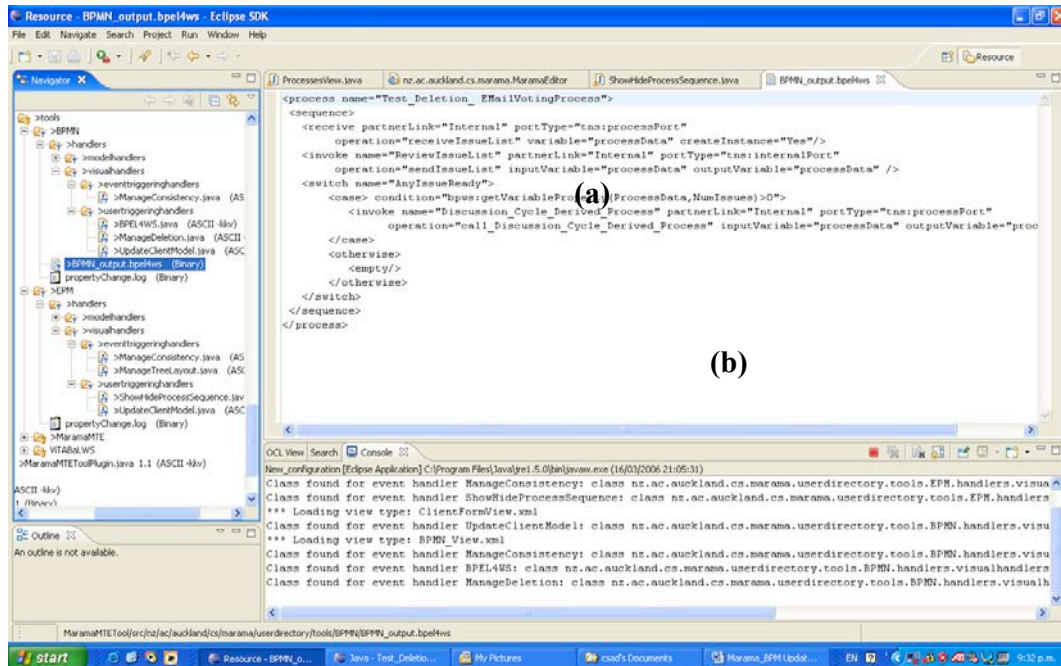
Form Chart diagrams could also be created in MaramaEML 1.0. Figure 6.11 shows an E-mail voting web submission example in a form chart diagram. Again, the main modelling diagram is shown in (a). The shape components and connectors are listed in (b) and (c). Detailed property information can be found and updated in (d).



**Figure 6.11 Form Chart Diagram in MaramaEML 1.0**

- BPEL code generation

Automatic BPEL code generation was implemented as an event handler in MaramaEML 1.0. This supports code generation directly from EML and BPMN diagrams (as shown in Figure 6.12). Detailed BPEL4WS code is represented in (a), and area (b) prints out a list of generation records for future debugging purposes.



**Figure 6.12: BPEL Code Generation in MaramaEML 1.0**

### 6.3.2 Brief Evaluation Process

At the beginning of the evaluation, EML and MaramaEML were briefly introduced to the participants and they were then asked to perform several predefined modelling tasks. The tasks were divided into three difficulty levels: simple, medium and complex. Participants were asked to repeat the same task in two different environments (pen and paper based EML modelling and software tool-based integrated EML and BPMN modelling). The whole process was monitored side by side and user was interviewed informally at the end of the evaluation.

### 6.3.3 Informal Evaluation Results

Feedback suggested EML and MaramaEML were very straightforward to use and understand. The users greatly favoured the tree overlay method for reducing the complexity of business processes compared to using only conventional BPMN views. They found it very valuable to have a tree overlay based modelling language as a supplement to overcome the shortcomings of existing business process notations. The multi-view collaboration is a useful approach to enhance the modelling strength. The

fish-eye zooming function is easy to use and significantly increases the navigation ability.

Several limitations and potential improvements were identified in our evaluations. These included:

- a need for more detailed mapping traceability
- a need to provide an integrated environment for the three modeling languages (EML, BPMN and Form Charts)
- a need to improve visual quality for EML and BPMN views
- a need to verify the BPEL code to guarantee its execution quality
- a need to enhance the scalability for MaramaEML (coping complex and large modeling diagrams)

#### **6.3.4 Improvements from the second evaluation**

Based on the evaluation feedback, we have developed MaramaEML 2.0, which addresses all the limitations identified in the previous section. These include:

- A text based log file has been created in the new system to enhance system traceability
- All three modeling views (EML, BPMN and FormChart) have been integrated into the same development environment
- A Labeled Transition System Analyser engine has been integrated into MaramaEML 2.0 to verify generated BPEL code. If the code passes the validation, it will provide an extra LTSA graphical view for the system
- New shapes and layouts have been developed to improve the visualisation quality of EML and BPMN
- Zooming and fish-eye view functions have been added to MaramaEML 2.0 to enhance visualisation scalability

#### **6.4 Large Formal Evaluation**

Following these improvements, EML 2.0 and MaramaEML 2.0 were more formally evaluated. It is challenging to perform this type of evaluation. Firstly, it is difficult to



establish a sufficient number of end users. An ideal evaluation would involve selecting a group of random users from a relevant population. However, in this real world, these kinds of end users are often busy. Secondly, evaluating a “big” environment such as MaramaEML is problematic. All we can hope to get is a general impression of task completion performance and qualitative impressions of usefulness and usability. The main objective is to prove that the concept is worth pursuing further.

#### **6.4.1 Participant recruitment**

Participants for this kind of formal evaluation are typically volunteers. Hence, strategies should be considered to increase to a maximum degree the recruitment response rate from the target user population. The following methods are typically applied in this kind of evaluation to enhance participant numbers (Dillon 2001):

- Minimize costs of completion. Questionnaires should be short, and easy and convenient to complete. Clear instructions should be given, together with completion guidelines, and key parts of the evaluation should provide definitions and additional help. Base on this guideline, the following work has been done:
  - Simple and clear questionnaires have been designed and reviewed. The questions are mostly mixed up with multiple-choice and diagram drawing questions.
  - A clear paper based instruction has been attached with the questionnaires.
  - A PDF version of instruction is also available online.
  - A short, face to face tutorial has been run at the beginning of the evaluation.
  - Some help functions have been designed in the software.
  - I stayed during the evaluation to answer potential questions.
- Maximize benefits. It is important to show to a potential target user that there are benefits for him or her to become involved in this evaluation. This can be

achieved by following a number of simple advices, such as showing positive regard, saying thank you, asking for advice, giving tangible rewards, etc. the following work has been done base on this guideline:

- A group thank-you E-mail has been sent to the potential end user group pre evaluation
  - A formal thank-you letter has been delivered to all the participants after the evaluation
  - I bought a MP3 player. All the evaluation participants have entered into a draw to win it.
- 
- Build trust via working with key influential people in organizations. People are more willing to accept an evaluation request if influential, managerial or other authoritative individuals comply with it. The following work has been done as per this guideline:
    - I have asked a wide range of academic staff in Computer Science and Software Engineering departments to help send the evaluation invitation.
    - I have used the Center for Software Innovation newsletter function to send out the evaluation invitation to all our industry partners
    - I have made several evaluation invitation presentations in the Computer Science and Software Engineering courses.
- 
- The initial contact letter (often via E-mail) stresses the usefulness of the evaluation to the organization, emphasized its importance to both research and practice and kindly ask the organization about their willingness to participate in the evaluation. The following methods have been performed as per this guideline:
    - A group E-mail has been sent out to all potential academic and industry participants (one month before the evaluation).
    - Follow-up E-mails and reminders have been sent out to the potential academic and industrial participants (one week before the evaluation).
    - A formal evaluation invitation has been published on front page of the Computer Science department website.

- A formal evaluation invitation has been published on the front page of the Electrical and Computer Engineering department website
- A formal evaluation invitation has been published on the front page of the Center for Software Innovation website

In summary, we have done every possible approach to increase the number of participants. The evaluation ran from August 2, 2007 ~ August 9, 2007. Over this one week period, a total of 38 end users agreed to take part. Eliminating 6 absences, we had a total of 32 usable participants covering Computer Science, Software Engineering and Business backgrounds.

#### **6.4.2 Evaluation Approach**

This evaluation involved both survey base and task performance. Detailed Formal Evaluation Schedule, Evaluation Questionnaire Example (Version 1, 2 & 3), Consent Form, University Ethics Approval Form and explanation sheets are attached as appendices.

- Survey Design

Survey design concerns the strategy for answering the questions or testing the hypotheses that stipulated the selection of the evaluation in the first place. It may be distinguished alongside three dimensions: time, unit of analysis and data analysis strategy (Hartson and Andre et al 2003).

- 1) Time: survey designs can either be cross-sectional or longitudinal, dependent on whether they exclude or include explicit attention to the time dimension. Given our evaluation does not contain special time consequences elements, a classical cross-sectional survey design is deemed appropriate.
- 2) Unit of Analysis: the unit of examination may be at an individual, group, departmental, social or organizational level. Alternatively, it may also involve an application system, portfolio, method, technique or other items from business elements. In our research, the unit of

analysis is the process modeler working with EML and BPMN process modeling languages.

3) Data Analysis Strategy: A wide variety of analysis approaches can be associated with survey data. We have adopted a value based data analysis approach (Rapide Design Team 1997) to gather the information for task completion, general quality feedback and usability quality attribute rate.

- Sampling Procedures

Sampling concerns the drawing of individual entities from a population of interest in such a way as permit generalization about the phenomena of interest from the sample to the population of concern. The most crucial element of a sampling procedure is the choice of the sample frame, which should be representative of the population. Our evaluation sampling procedure is based on group separation quota sampling and systematic sampling strategies to guarantee data quality.

- Data Collection

The choice of a data collection method, such as mail questionnaire, telephone interview, face-to-face interview or web based survey is significant because it has impacts on the quality and cost of the data collected. In order to guarantee the data's correctness, we decide to use face-to-face interviews and side-by-side operation observation to gather first-hand information for the final analysis.

### **6.4.3 Brief Evaluation Schedule**

The whole evaluation took approximately two hours in total. The end users were separated into different groups to evaluate the usefulness of the visual language and support environment. This comprised eleven steps:

**Step 1: Evaluation Schedule Introduction**

**Step 2: *EML & BPMN Introduction***

Participants were briefly introduced to EML (Enterprise Modelling Language) and BPMN (Business Process Modeling Notation), and some working examples were explained (including a Travel Booking system)

**Step 3: *nDeva++ Introduction***

The target modeling example was introduced (nDeva++, a web-based University Enrolment System)

**Step 4:** Download EML Tool, User Guide and EML introduction slides from the website

**Step 5: *Modelling Task 1***

Participants were divided into two groups.

Group 1 is asked to model nDeva++ overall structure using EML

Group 2 is asked to model nDeva++ system use BPMN

**Step 6: *Modelling Task 2***

Group 1 Participants are asked to add an “Enrol in a Paper” task process using EML

Group 2 Participants are asked to add an “Enrol in a Paper” task process using BPMN

**Step 7: *Modelling Task 3***

All the participants are asked to save their work in the computer and use the other language to repeat Step 5 ~ 6 again (group 1 participants now use BPMN, and group 2 EML).

**Step 8: *Modelling Task 4***

All the participants were asked to try show / hide tasks and trigger functions, show / hide tree component functions.

**Step 9: Modelling Task 5** (10 minutes)

Participants were asked to try the fisheye view and zooming functions, code generation and validation functions and explore other support functions in MaramaEML

**Step 10: Answer the Questionnaire** (30 minutes)

Participants were mixed and then divided into three new groups. Group one was asked to complete Questionnaire 1 (General Usability Abilities Questionnaire) and group two & three were required to answer Questionnaire 2 & 3 (Cognitive Dimensions Walkthrough Questions)

**Step 11: Submit the Consent Forms and Questionnaires**

**6.4.4 Data Analysis**

- Participant Backgrounds

<b>IT Background</b>	<b>Percentage</b>
No IT background	<b>5%</b>
Job training IT experience	<b>5%</b>
Formal Computer Science education background	<b>35%</b>
Formal Software Engineering education background	<b>45%</b>
Other	<b>5%</b>
Unspecified	<b>5%</b>

**Table 6.1: Participants' IT Background Distribution**

<b>Business Process Modeling Background</b>	<b>Percentage</b>
Know BPM well	<b>5%</b>
Does not have BPM experience	<b>30%</b>
Self-learnt BPM techniques	<b>60%</b>
Other	<b>5%</b>

**Table 6.2: Participants' BPM Background Distribution**

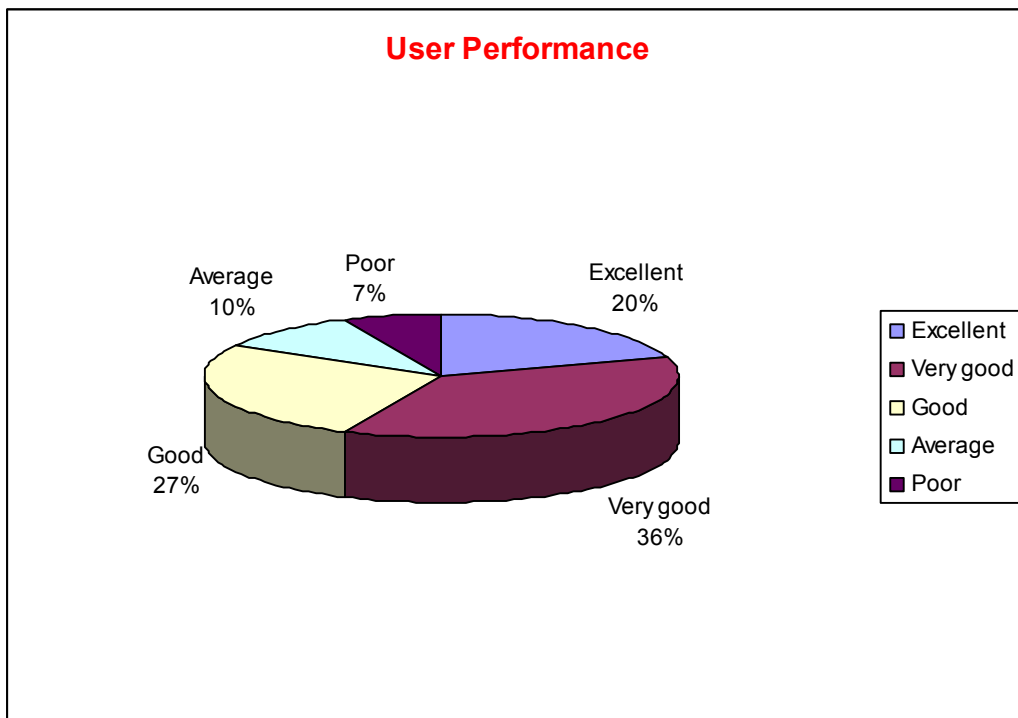
The user background coverage for this evaluation was reasonable broad (as shown in Table 6.1 and Table 6.2). We had 10% of the participants who didn't have any formal IT background, however 5% of them had some informal job related IT training experiences. Nearly 30% didn't have BPM process experiences. 60% learned BPM by themselves. We also have 80% IT experts, 35% are from Computer Science and 45% Software Engineering. 5% didn't specify their IT background, and 5% of comes from other backgrounds.

- Task Completion

Overall, 97% of the participants completed their tasks in 2 hours' time. Considering 30% of them don't have business process modeling experience, this is a very successful result. However, 3% of the participants could not complete their tasks or gave up during the evaluation.

- User Performance

Figure 6.13 shows the user performance analysis results. We have collect all the answers from the participants and compare them with the model answers. 20% of the participants did an "Excellent" job using software and EML to complete the tasks. 36% were rated "Very Good" and 27% "Good". This leads to an 83% fine performance rating in total (by using MaramaEML and EML). 7% had poor quality answersr, and 10% at an average level.



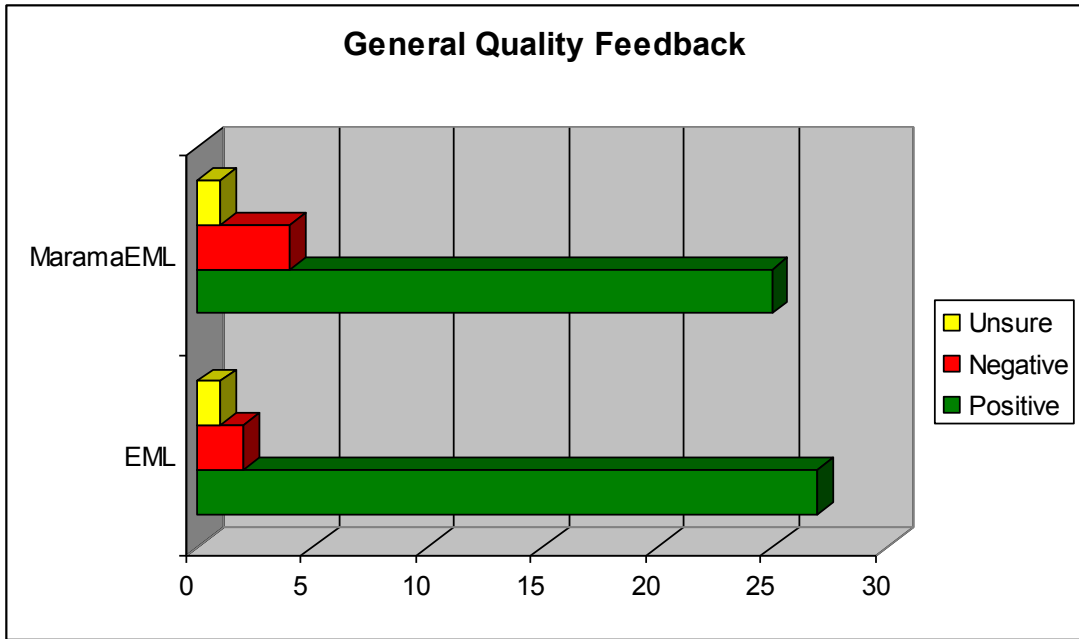
**Figure 6.13 User Performance Diagram**

- General quality for EML and MaramaEML

We had very positive feedback on the perceived quality of the language and support tool (as shown in Figure 6.14). 27 out of 32 participants think the overall quality of EML is positive. 3 out of 32 think it negative and 2 were not sure.

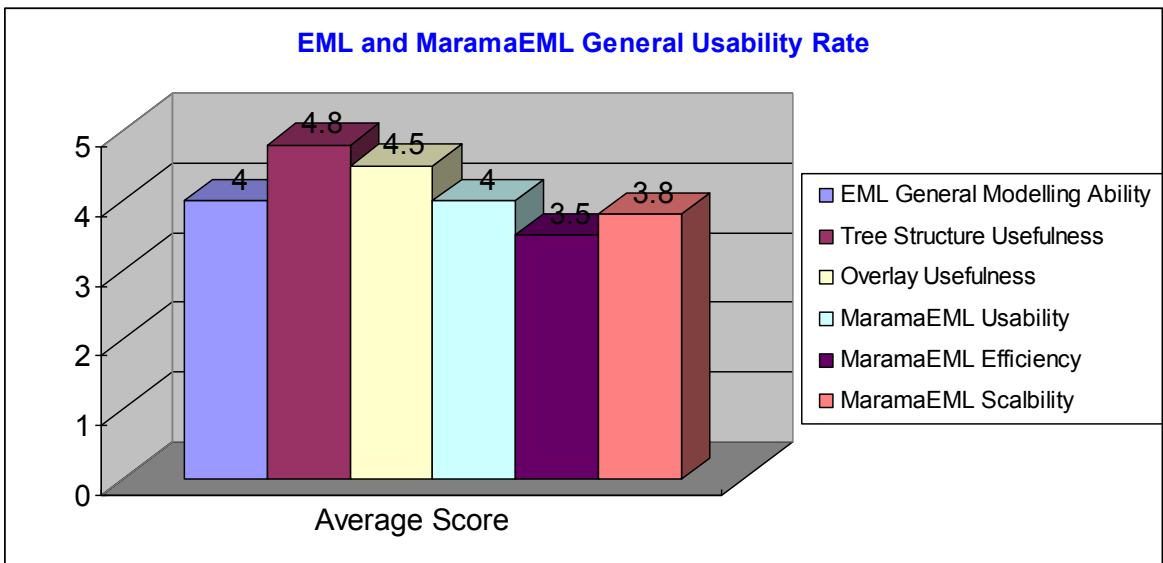
For the software tool, 24 out of 32 participants were positive about the general quality. 6 out of 32 were negative; and 2 were unsure.





**Figure 6.14 General Quality Feedback Diagram**

- General usability data



**Figure 6.15: EML and MaramaEML Usability Rate Summary**

The usability results for EML and MaramaEML are also promising. Figure 6.15 shows averaged result summaries from our questionnaires. The “EML Tree Structure Usefulness” received the highest average score (4.8 out of 5). Most participants believe that using a tree overlay structure as a modeling approach will strongly

enhance the modeling ability. The “overlay usefulness” and “EML General Modeling Ability” received averages of 4.5 and 4 out of 5 respectively. However, the two lowest results come from MaramaEML’s Scalability and Efficiency. The problem was caused by the unresponsive computing speed of the software tool (when zooming a large, complex diagram), and the need for better control of information hiding.

#### **6.4.5 Improvements from the formal evaluation**

The latest version of MaramaEML has been released (version 3.0) with improvements based on the suggestions from the 3<sup>rd</sup> round of the formal evaluation. An improved zooming and fisheye view function has been built in it to enhance the scalability.

### **6.5 Summary**

In this chapter, we have introduced three rounds of evaluations of EML and MaramaEML. Each round had a different testing focus and the results in each case have been used to improve the overall quality of EML and MaramaEML.

A positive outcome from conducting these cycles of evaluations and improvements is that MaramaEML 3.0 has been demonstrated at the 23rd IEEE/ACM Automated Software Engineering Conference in L’Aquila, Italy (ASE 2008), and received the “Best Demo Award” from that conference.

# Chapter 7

## CONCLUSION

---

The overall aim of the research presented in this thesis was the design and implementation of Enterprise Modelling Language (EML) and its software support tool (MaramaEML). During the research a wide range of visual modeling methods, their visual notations and the support software have been reviewed and compared for the strengths and weaknesses. Bases on the findings, we have designed a novel business process modeling notation using tree overlay structure. An Eclipse-based software tool has been developed to support the modeling capability for EML, and provide extra integration functions. Three rounds of evaluations have been conducted during the development to inform and refine the design.

### 7.1 Research Output

All the research targets explained in Chapter 1 have been achieved. The design of a novel tree overlay based visual modeling language (EML) has been completed (please refer to Chapter 3 for detailed information). It has successfully mitigated the complexity issue and cobweb problem in traditional flow chart based diagrams. The multi-layer structure of the language provides a flexible solution to support both organizational and process level views for the system. Majority of the users find the new language easy to learn and use, without requiring a deep learning curve or formal programming background (feedback gathered from the evaluation). The language has a strong modeling ability for enterprise level business process services, processes and sub-processes, operations, exception handler, as well as some advanced constructs such as dependency, trigger, iteration and condition.

The software tool for EML has been developed (MaramaEML). Please refer to Chapter 4 for detailed information about the tool implementation. The software tool supports creating, inspection, editing and storage of EML. It also provides the ability

to generate BPEL code from EML models automatically. A LTSA engine has been integrated into MaramaEML for the BPEL code validation. The software tool also integrates a BPMN view, and it has the multi-view support function for both BPMN and EML. Being Eclipse based, and with the ability to generate BPEL code, MaramaEML has strong potentials to integrate third party tools, especially Eclipse based software environments. Together with the EML's multi-overlay and elision functionalities to reduce the complexity, MaramaEML also includes a distortion based fisheye view and a zooming view to enhance the overall scalability.

Three rounds of evaluations have been carried out (Cognitive Dimensions Walkthrough, informal domain expert evaluation and formal large end user evaluation). Please refer to Chapter 6 for detailed evaluation information. We have continuously used the results from these evaluations to refine the functionality, usability and other related quality attributes of the EML and MaramaEML.

A list of refereed publications has been generated during the research (please refer to Chapter 1.4 for detailed information). The work has received "Best Software Tool Demo Award" at ASE 08, in Italy, and two of the papers have been nominated as "Best Research Paper of the year" in the Department of Computer Science, the University of Auckland.

The EML and MaramaEML have been included as formal lecture materials (as a good example of business process modeling notation and software tool) for a fourth year Software Engineering course (at Electrical Computer Engineering Department) and a first year postgraduate Computer Science course (at Computer Science Department) at the University of Auckland.

The EML and MaramaEML have been applied in real business consulting work for Sofismo Limited ([www.sofismo.ch/](http://www.sofismo.ch/)), a Switzerland ICT consulting firm (for their finance and banking modeling project).

## 7.2 Future Work

Several improvements and extensions could be made to the EML and MaramaEML to increase their flexibility and functionality. These include:

- **Extra UML View Support:** During our end user evaluation, we have received strong demands for adding a UML view into the framework. An additional UML view can be added in MaramaEML via the multi-view support.
- **Consistency Checking:** The current version of MaramaEML includes a basic consistency checking method between EML views and BPMN views. We are using an event log to keep the usage records and trace the model changes. If the user changes the name of a model element in an EML view, an event handler will check the corresponding mapped notation in a BPMN view and update the name automatically. However, this approach cannot cope with the situation requiring complex logical analysis. For example, if the user tries to delete a node which has several sub-nodes in EML, the system at this stage will automatically delete all the sub-nodes in an EML view and all the corresponding nodes in a BPMN view. This solution is insufficient. In the future, we need to incorporate a more comprehensive way to handle this in MaramaEML (e.g. link the sub-nodes with other nodes if there is an underlay relationship between them). We also want to make use of the OCL-based technique provided in Marama meta-tools (Li, 2007) to define dependency and consistency constraints in the EML meta-model.
- **Special Version Notation for Color Blind User:** During the end user evaluation, we observed that an achromatopsia participant became totally lost in the overlay integration view. An overlay integration view normally mix process flows (blue color), trigger flows (red color) and exception flows (green color) in the same view. With this version of EML, color is a very important design component to represent and distinguish the information. However, from the evaluation, we found this important piece of visual information disappeared when the notation was viewed by a color blind user.

We have done some research after the evaluation. We found that there are four common types of color people (partially sighted, dyslexia, hearing impaired and physically impaired), so there is no single and simple solution to amend this in a short period. Future research is required to design a special version of EML for this group of users.

- **Improve Software Performance:** During the end user evaluation, we found that some users had unresponsive issues with the tool when they were using the zooming or fisheye view function for large EML trees. However, the users with higher performance computers did not have this issue. In the future, an improved zooming and fisheye view algorithm is required to improve the performance.
  
- **Integration with other Software Tools in the Research Group:** Some very good software tools have been developed in my research group (all based on the Marama framework). They focus on different areas other than business process modeling. We are working on possible solutions to integrate our software tool from the high level business process modeling domain to their architecture or application level domains. Examples are MaramaMTE and ViTABal-WS.
  - MaramaMTE (Middleware Testing Environment) is a tool for modelling complex software architectures and generating performance test beds from these models to assess likely software architecture performance (Cai and Grundy et al 2007). Figure 7.1 shows an architecture modelling example with MaramaMTE architecture view. We are exploring the possibility to integrate software architecture specification and performance modelling with MaramaMTE via the Marama platform.

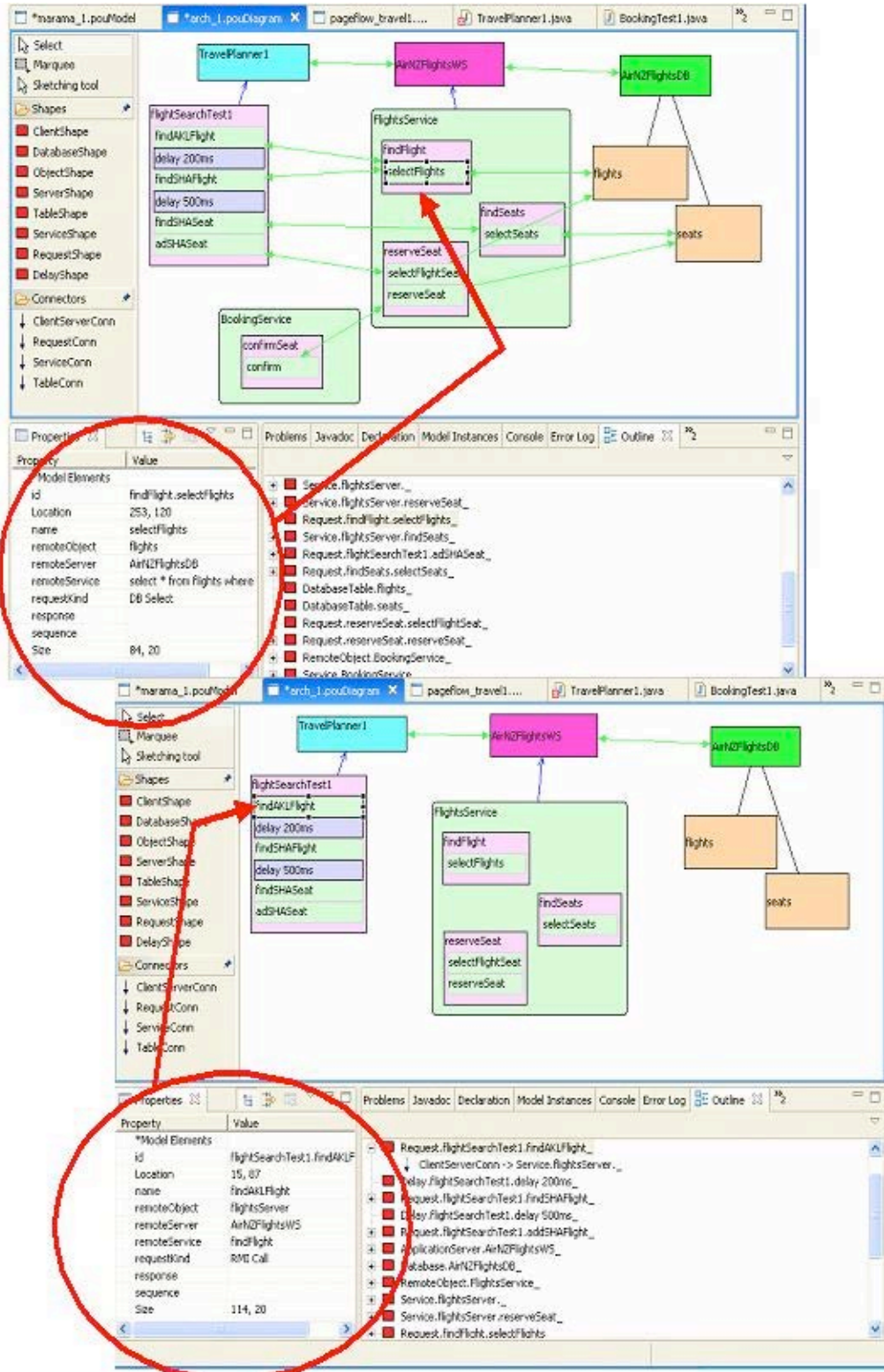
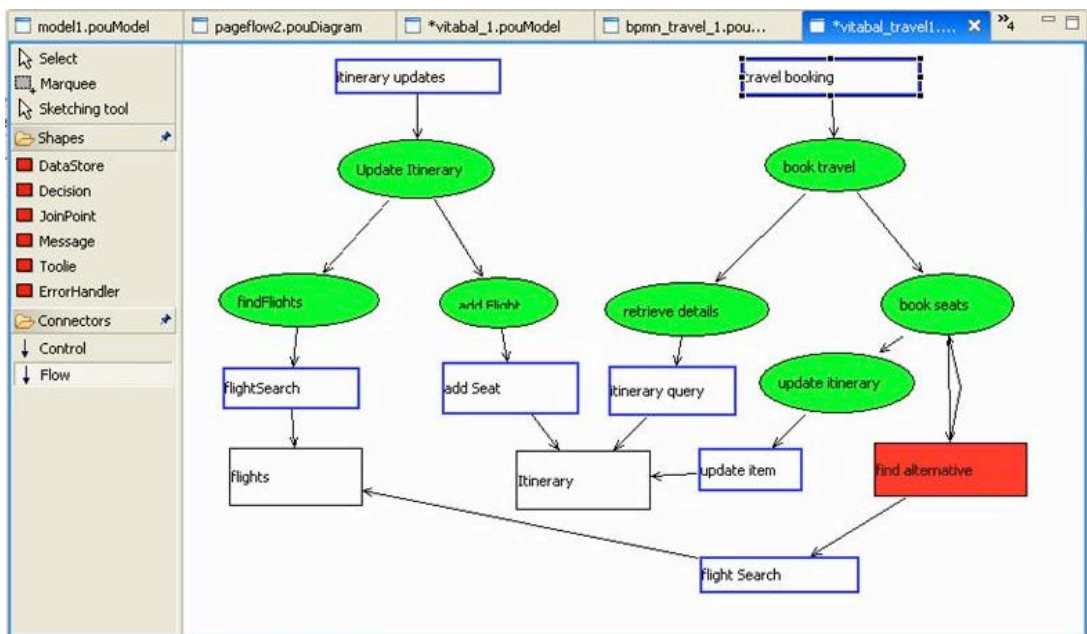


Figure 7.1: MaramaMTE Architecture View

- ViTABaL (Grundy and Hosking 1995) is a hybrid visual programming environment for designing and implementing event based systems. It uses the Tool Abstraction paradigm to compose systems by integrating, and coordinating toolies and abstract data structure components. ViTABaL-WS, which specializes the ViTABaL visual composition language to the domain of web services composition, supports modelling of both event-dependency and dataflow in designing complex web service compositions. Figure 7.2 shows an example modeling web service composition using Marama-ViTABaL-WS. We are looking to integrate via the Marama framework with ViTABaL-WS to extend our functionality from business process modeling domain to web service specification domain.



**Figure 7.2: Web Service Composition in ViTABaL-WS**

### 7.3 Conclusion

To the best of our knowledge, EML is the first tree overlay structure based visual language in the area of business process modeling. EML is a novel business process modelling language based on tree hierarchy and overlay metaphors. Complex business architectures are represented as service trees and business processes are modelled as



process overlay sequences on the service trees. By combining these two mechanisms EML gives the user a clear overview of a whole enterprise system with business processes modelled by overlays on the same view. An integrated support tool for EML has been developed using the Eclipse based Marama framework. It integrates EML with existing business notations (e.g. BPMN) to provide high-level business service modelling. A distortion-based fisheye zooming function enhances complex diagram navigation ability. MaramaEML can also generate BPEL code automatically from the graphical representations and map it to LTSAs for validation.

These support functions in MaramaEML, together with the strong modeling capability of the Enterprise Modeling Language, offer a good way forward for the users of the business process modeling domain.

# Appendix A

## RESEARCH PROJECT APPLICATION FORM

---

### Important Information

- Applications will only be accepted on forms dated for the current year.
- Please complete this form in reference to the [UAHPEC Users' Guide 2007](#), and **Frequently Asked Questions (FAQs)** available on the University of Auckland website under Research and Ethics and Biological Safety Administration.
- Submit one **unstapled, single sided, signed** copy of the form and all accompanying documentation to the Research Ethics and Biological Safety Administration, Room 005, Alfred Nathan House, 24 Princes Street.
- All Participant Information Sheets (PISs) and Consent Forms (CFs) must be submitted on University of Auckland departmental letterhead which has the full contact address for the department. These may be on electronic letterheads. Letterheads are available from the applicant's department.
- **Note:** Some faculties have an earlier closing date for the agenda as there are special requirements, for example, the Faculty of Medicine and Health Sciences requires a signed Dean's Signature Sheet, the Faculty of Education has an earlier closing date as it requires Ethics Advisor sign off.
- Applicants will receive an email acceptance letter with the reference number included. **It is essential to quote this reference number with all communication to UAHPEC and participants, in their PISs and CFs.**



Reference Number.....

**University of Auckland Human Participants Ethics Committee (UAHPEC)  
RESEARCH PROJECT APPLICATION FORM (2007)**

**GENERAL INFORMATION**

1. **PROJECT TITLE:** A Visual Language and Support Tool for Business Process Modeling

2. **APPLICANT/PRINCIPAL INVESTIGATOR (P.I.)**

(This will be the supervisor for a Masters student. Doctoral students submit in their own names but the Supervisors must sign the form.)

**Name:** Lei Li

**Address:** Department of Computer Science

**Email address:** l.li@cs.auckland.ac.nz

**Phone number:** 3737599 ext. 82128

3. **NAME OF STUDENT:**

(If applicable)

**Email address:**

**Phone number:**

**Name of degree and Department:**

4. **OTHER INVESTIGATORS:**

**Names:**

**Organisation:**

**Is ethical approval being applied for from another institution?**

**NO**

(If YES, indicate name of the institution and attach evidence.) .....

5. **AUTHORISING SIGNATURES:**

**HEAD OF DEPARTMENT:** ..... **Date:** .....

**HOD name printed:** ..... **Department:** .....

6. **APPLICANT'S DECLARATION**

The information supplied is, to the best of my knowledge and belief, accurate. I have read the current University of Auckland Human Participants Ethics Committee User's Guide 2007. I clearly understand my obligations and the rights of the participants, particularly in regard to obtaining freely given informed consent.

**Signature of P.I. /Supervisor:**..... **Date:** .....

**Signature of Student:**..... **Date:** .....

If a student project, including doctorate, signatures of both the Supervisor and the student are required.

## SECTION A: PROJECT

### 1. AIM OF PROJECT:

#### a) What is the hypothesis / research question(s)? (State briefly)

Diagrams that model business processes can become very complex and therefore difficult to interpret. Our hypothesis is that by overlaying the model with a tree structure and providing appropriate software support that the presentations are less complex and therefore easier to interpret.

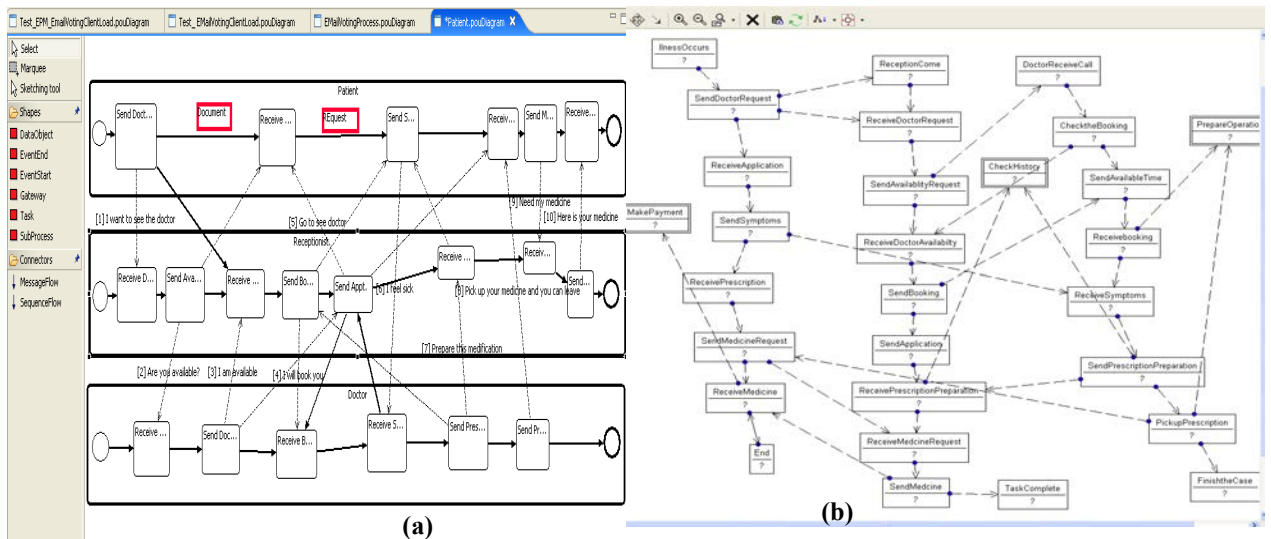
#### b) What are the specific aims of the project?

The goal of this project is to explore a tree overlay structure-based diagrammatic approach to business process modeling. These diagrams may ease communication with end users while retaining rich expressiveness. There is a usability testing component part to the project which requires human participants, for which ethical approval is sought.

### 2. RESEARCH BACKGROUND

Provide sufficient information to place the project in perspective and to allow the significance of the project to be assessed.

Visual business process modeling fulfils an important role to enable high-level specification of system interactions, improve system integration and support performance analysis. Since the early 1970s many languages, standards, methodologies and tools for enterprise modeling have been created. Examples include Entity-relationship models, Data Flow Diagrams, Flow Charts, Scenarios, Use Cases, and Integration Definition for Functional and workflow modeling.

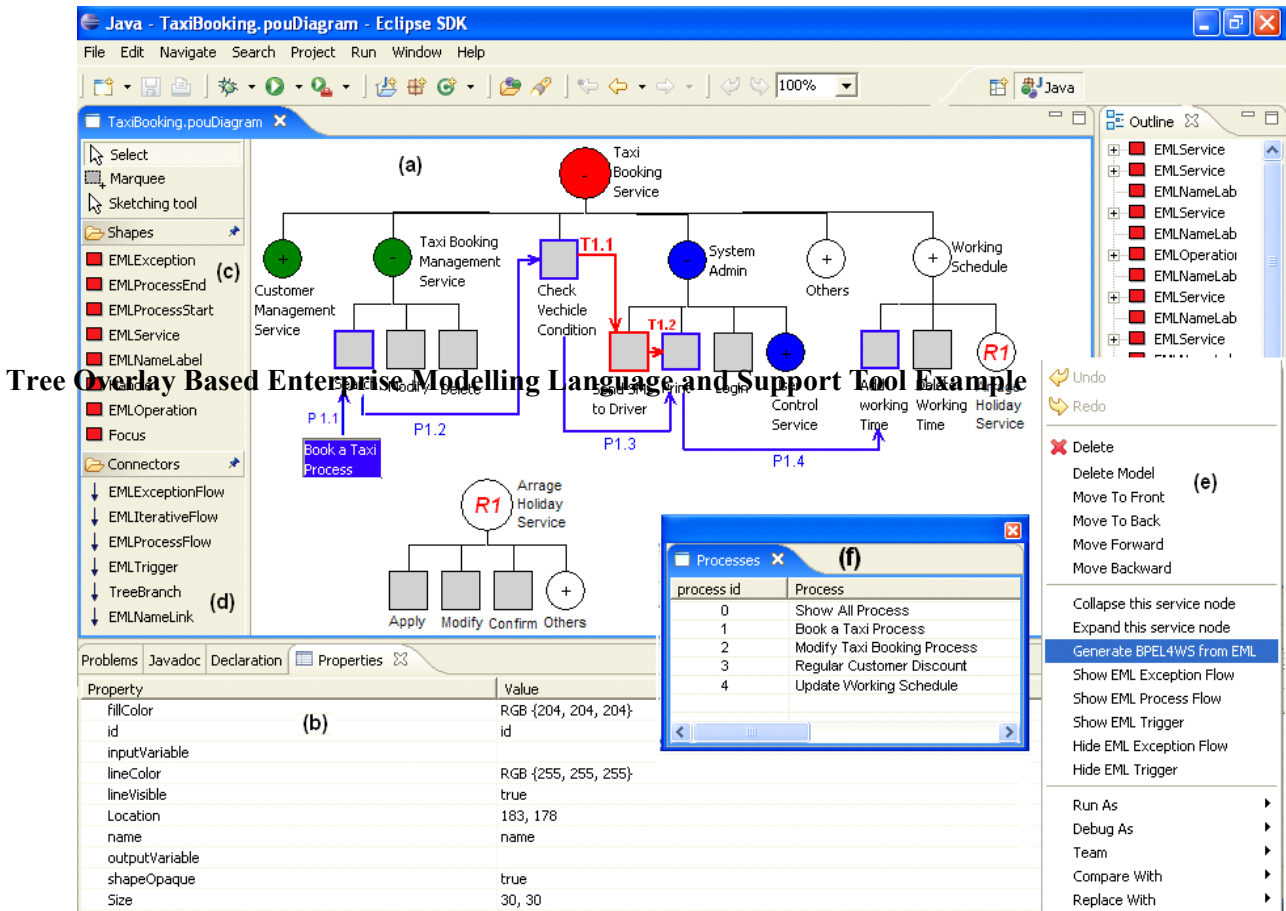


Traditional workflow based modeling language example

However, a common difficulty in all of these approaches is an appropriate visual method to reduce the complexity of large diagrams. Most existing modeling technologies are effective in only limited problem domains or have major weaknesses when applied to large systems models e.g. “cobweb” and “labyrinth” problems (with users having to deal with many cross diagram flows). This raises significant scalability issues: the simple example presented in the above figure demonstrates this kind of problem. Software tools used to create these models employ multi-view and multi-level approaches to mitigate this problem. These approaches have achieved some

success but do not fully solve the problem because using the same notation and flow method in a multi-view environment just reduces individual diagram complexity, but increases hidden dependencies. It requires long term memory of the users, as they have to build and retain the mappings between views mentally. In addition, most existing flow based business modelling notations lack multiple levels of abstraction support.

In contrast, using a tree structure is an efficient way of representing the hierarchical nature of complex systems graphically. Trees also support navigation, elision and automatic layout in ways difficult to achieve with current graph-based approaches. We chose to use trees as they are familiar abstractions for managing complex hierarchical data; can be easily collapsed and expanded to provide scalability; can be rapidly navigated; and can be over-laid by cross-cutting flows and concern representations. Our earlier work on modelling complex user interfaces and their behaviour with tree-based overlays demonstrated these benefits. We have designed EML (Enterprise Modelling Language), a novel tree overlay-based visual notation and its integrated support environment (MaramaEML) to supplement and integrate with existing enterprise level modelling solutions.



Above figure shows a simple example of an EML tree structure modelling a composite taxi booking service. The customer management, taxi management, system admin and working schedule services are sub-services (represented as ovals) of the taxi booking service. The system admin service also includes an embedded user control service. The rectangle shapes represent atomic operations inside the service. In an EML-modelled enterprise system, major services are represented as separate trees. Each business process is represented as an overlay on the basic tree structure or an orchestration between different service trees. In a process layer, users have the choice to display a single process or collaboration of multiple processes. By modelling a business process as an overlay on the service tree

structure, the designer is given a clear overview of both the system architecture and the process at the same time. Processes can be elided mitigating the cobweb problem common in existing flow based visual notations.

We have developed an integrated design environment (MaramaEML) for creating EML specifications. MaramaEML aims to provide a platform for efficiently producing EML visual models and to facilitate their creation, display, editing, storage, code generation and integration with other diagrams. Above figure shows a screen dump of a MaramaEML model in use with a typical EML tree with a process overlay. The user produces a *Book a Taxi* process in (a) using the MaramaEML modeling diagram tools. To the left of the EML diagram area are the MaramaEML shapes (c) and connectors (d) toolbars. This provides options relating directly to the construction and editing of EML tree in the central work area (a). The EML process layer is then compiled to BPEL4WS executable code via code generation handler in (e). Code is generated by model dependency analysis and translation to structured activity constructs. MaramaEML aims to provide a platform for the efficient production and navigation of EML. The tool supports a drag and drop approach and any parts of an EML tree can be directly selected and moved. Elision and expansion are triggered via popup menu (e).

As a short summary, the highlight of our research is its flexibility in modelling business processes using different layers. A service-oriented tree structure represents the system functional architecture. Business process modelling is constructed as an overlay on top of this service tree. By using a multi-layer structure, an enterprise system can be modelled with a variety of early aspects to satisfy design requirements. An Eclipse based software tool, MaramaEML has been developed to edit EML diagrams integrated with existing modelling languages such as BPMN (Business Process Modelling Notation) and supports automatic generation of industry communication standard ---- BPEL (Business Process Execution Language) code.

Business process modeling presents unique challenges that a visual language approach may successfully address particularly issues related to visual methodology overheads between information technology experts and end users. We have had a foundational paper on this work accepted for presentation at the 9<sup>th</sup> ACM International Conference on Enterprise Information Systems to be held in June 2007 at Portugal. The work proposed here extends that foundational work to include a full usability study requiring human participants, for which we seek ethical approval.

The proposed usability study will evaluate the effectiveness of this new visual model in practice. Users' experience will be recorded in a questionnaire form. All participants will be normal adults who are information technology or business modelling literate.

- 3. Describe and discuss the ethical issue(s) arising from this project.** (UAHPEC expects applicants to identify the ethical issues in the project and explain in the documentation how they have been resolved. A "Not Applicable" response is not acceptable. The application will not be considered if this is not answered adequately.)

We do not foresee any potential ethical issues that could arise from the usability testing. The applicant is the member of an academic department (Computer Science). He has no influence on potential participants' academic outcomes in business process modelling related courses as no such courses are run in the Computer Science Department. Also there will be no tangible financial rewards for participants as they will be take part in the usability testing on a strictly voluntary basis.

## **SECTION B: PARTICIPANTS**

- 1. What types of people are participating in the research?** (Delete those who do not apply).

**Normal Adults**

2. **Explain how many organisations, or departments within the organisations, and individuals you wish to recruit.** (Attach any letter of support you may have had from an organisation.)

Maximum 30 participants.

The user testing will be conducted by a single person (the applicant) over a month with each session consisting of a maximum of 4 persons using the software tool. The participant size has been chosen to provide an optimal environment for group interactions considering the limited human resources available.

3. **How will you identify your potential participants?** (If by advertisement or notice, attach a copy)

Industry business modelling specialists, computer science and software engineering students will be invited to participate. A participant information sheet for *Participant Recruitment* which will be used in recruiting participants is attached. The content of the information sheet will be used for public notices and emails.

4. **How and where will potential participants be approached? Explain how you will obtain the names and contacts of participants.** (For example by email, by advertisement, through an agency holding these details.)

Email to selected industry representatives and graduate students. We plan to invite around 100 people to participate, selecting up to 30 on a first-come basis.

5. **Who will make the initial approach to potential participants?** (For example, will the owner of the database send out letters on behalf of the researcher?)

By the applicant, Mr Lei Li

6. **Is there any special relationship between participants and researchers?** (For example, student or teacher. If YES, explain.)

NO

7. **Are there any potential participants who will be excluded?**  
(If YES, explain, and state the criteria for excluding participants.)

YES

Due to the limited resources, we will recruit no more than 30 participants. This will be done in a strictly first come, first served basis. Only business process modelling specialists or computer science and software engineering literate applicants will be recruited, as these are the target end-users of the software tool under study.

## **SECTION C: RESEARCH PROCEDURES**

There is a need here to fully inform the Committee about all factors relating to the research including, where appropriate, the researcher's qualifications to conduct this work (investigation).

1. **PROJECT DURATION** (approximate dates): **From 1/Oct/2003 to 30/Sep/2010**

2. **Describe the study design.** (For example, longitudinal study)

- i) Usability study
- ii) Questionnaire

3. **List all the methods used for obtaining information.** (Attach questionnaires, research instruments, interview schedule to this application).

I am going to observe, take notes and collect changed diagrams. *(maybe video or something)???*

**4. Who will carry out the research procedures?**

By the applicant (Lei Li) and his supervisors (Prof. John Hosking and Prof. John Grundy)

**5. a) Where will the research procedures take place?** (Physical location or setting.)

Computer Science laboratory, The University of Auckland; or  
Industry participants' company (if required)

**b) If the study is based overseas, which countries are involved?** (Provide local contact information on the PIS(s).)

N/A

**c) If the study is based overseas, explain what special circumstances arise and how they will be dealt with? Explain any special requirements of the country and / or the community with which the research will be carried out.**

N/A

**6. How much time will participants need to give to the research?** (Indicate this in the PIS.)

The usability study will observe the participants undertaking their given tasks. At the end of the testing session the evaluation questionnaires will be filled by the participants. The whole procedure will take no more than two hours.

**7. Does this research include the use of a questionnaire / email?** (If YES, attach a copy to this application.)

**YES**

**8. Are you intending to conduct the research in (University) class time?** (If YES, include advice from the Course Coordinator giving approval for this to occur.)

**NO**

**9. Is deception involved at any stage of the research?** (If YES, justify its use, and describe the debriefing procedure.)

**NO**

**10. Will information on the participants be obtained from third parties?** (For example, from participant's employer, teacher, doctor etc. If YES, explain, and indicate in the PIS(s).)

**NO**

**11. Will any identifiable information on the participants be given to third parties?** (If YES, explain, and indicate in the PIS.)

**NO**

**12. Provide details on any compensation or reimbursement of expenses, and where applicable, level of payment to be made to participants.** (If payment/koha is offered, explain in the PIS.)

All participants are volunteers and they will not be rewarded financially for their participation.



13. a) **Does the research involve the administration of any substance** (For example, eye-drops or food) **to participants?**

NO

- b) **Does this research involve potentially hazardous substances?** (For example, radioactive materials)

NO

## SECTION D: INFORMATION AND CONSENT

1. **By whom and how, will information about the research be given to participants?** (For example, in writing or verbally – a copy of the information given to prospective participants in the form of a PIS must be attached to this application.)

Verbally and in writing, *Participant Information Sheet for Usability Testing* is attached.

2. a) **Will the participants have difficulty giving informed consent on their own behalf?** (Consider physical or mental condition, age, language, legal status, or other barriers.)

NO

- b) **If participants are not competent to give fully informed consent, who will consent on their behalf?** (For example, parents / guardians)

N/A

3. **Consent should be obtained in writing. Explain and justify any alternative to written consent.**

Consent will be obtained verbally and in writing, the latter using the attached consent form.

4. **UAHPEC requires that access to the Consent Forms be restricted to the researcher and/or the Principal Investigator. Confirm that you intend to do this otherwise, please explain.**

YES

5. **Will Consent Forms be stored by the Principal Investigator, in a locked cabinet, on University premises?**

YES

6. **It is required that Consent Forms be stored separately from data and kept for six years. Confirm that you intend to do this otherwise please explain.**

YES

## SECTION E: STORAGE AND USE OF RESULTS

1. **Will the participants be audio-taped, video-taped, or recorded by any other electronic means?** (If YES, explain in the PIS and the CF. Consider whether recording is an optional or necessary part of the research design, and reflect this in the CF.)

NO

2. a) **How will data, including audio, videotapes and electronic data be handled and stored to protect against unauthorised access?** (Explain this in the PIS with details of storage, possible future use and eventual destruction.)

All resulting observational data (questionnaire feedbacks) will be compiled and will be kept in a secure locked cabinet within the Department of Computer Science, the University of Auckland.

- b) **If the tapes are being transcribed or translated by someone other than the researcher, explain what arrangements are in place to protect the confidentiality of participants.** (Attach any Confidentiality Agreements to this application.)

N/A

- c) **If recordings are made, will participants be offered the opportunity to edit the transcripts of the recordings?** (In either case, the PIS must inform the participants. Where participants are asked to make a choice, this should be shown on the CF.)

No recordings will be made.

- d) **Will participants be offered their tapes (or a copy thereof)?** (In either case, the PIS must inform the participants. Where participants are asked to make a choice, this should be shown on the CF.)

No recordings will be made

- e) **Will data or other information be stored for later use?**

NO

**If YES, explain how long the data will be stored and how it will be used.** Indicate this in the PIS. The period data is to be kept will be commensurate to the scale of the research. For peer reviewed publication or research that might be further developed, the UAHPEC expects six years.

- f) **Describe any arrangements to make results available to participants, including whether they will be offered their tapes.** Explain this in the PIS. Where participants are asked to make a choice, this should be shown on the CF.

A summary of the survey results and resulting publications will be made available to participants on request.

3. a) **Are you going to use the names of the research participants in any publication or report about the research?** The PIS must inform the participants, and be part of the consent obtained in the CF.

NO

- b) **If you don't use their names, is there any possibility that individuals or groups could be identified in the final publication or report?** If YES, explain, and describe in the PIS. This is a problem either when one is dealing with a small group of participants known to a wider public or when there is to be a report back to participants likely to know each other.

NO

## SECTION F: TREATY OF WAITANGI

1. **Does the proposed research impact on Māori persons as Māori? If YES, complete all questions in this section and attach evidence of consultation from the nominated Māori Advisor within your Faculty.** If NO, go to Section G.

NO

2. **Explain how the intended research process is consistent with the provisions of the Treaty of Waitangi.** Refer to the User's Guide 2007 for further information.
3. **Identify the group(s) with whom consultation has taken place, describe the consultation process, and attach evidence of the support of the group(s).**
4. **Describe any on-going involvement the group(s) consulted has / have in the project.**
5. **Describe how at the end of the project information will be disseminated to participants and the group(s) consulted at the end of the project.**

## **SECTION G: OTHER CULTURAL ISSUES**

1. **Are there any aspects of the research that might raise any specific cultural issues, other than those covered in Section F?** If YES, explain. Otherwise go to Section H.

NO

2. **What ethnic or cultural group(s) does the research involve?**
3. **Identify the group(s) with whom consultation has taken place, describe the consultation process, and attach evidence of the support of the group(s).**
4. **Describe any on-going involvement the group(s) consulted has / have in the project.**
5. **Describe how information will be disseminated to participants and the group(s) consulted at the end of the project.**

## **SECTION H: CLINICAL TRIALS**

1. **Is this project a Clinical Trial?** (If YES, complete section, otherwise go to Section I. If YES, attach ACC Form A or B – see Guidelines)

NO

2. **Is this project initiated by a Pharmaceutical Company?**

NO

3. **Are there other NZ or International Centres involved?**

NO

4. **Is there a clear statement about indemnity?**

N/A

5. **Is Standing Committee on Therapeutic Trials (SCOTT) approval required?**

N/A

6. **Is National Radiation Laboratory approval required?** (Attach)

N/A

7. **Is Gene Therapy Advisory Committee on Assisted Human Reproduction (NACHDSE) approval required?**

N/A

## **SECTION I: RISKS AND BENEFITS**

1. **What are the possible benefits to research participants of taking part in the research?**

Participating in this study may increase their understanding of issues related to and knowledge of visual languages in the context of a business process modelling.

For industry business process modelling professionals, participating in this study exposes them to current research in the field and provides an introduction to the university research team.

2. **What are the possible risks to research participants of taking part in the research?** Make sure that you have clearly identified or explained these risks in the PIS.

Nil

3. a) **Are the participants likely to experience discomfort (physical, psychological, social) or incapacity as a result of the procedures?** If YES, describe, and explain them clearly in the PIS(s).

NO

b) **What other risks are there?**

Nil

c) **What qualified personnel will be available to deal with adverse consequences or physical or psychological risks?** Explain in the PIS.

N/A

## **SECTION J: FUNDING**

1. **Do you have or intend to apply for funding for this project?** If YES, complete this section and acknowledge it in the PIS, otherwise proceed to Section J.

NO

2. **From which funding bodies?**

N/A

3. **Is this a UniServices Ltd project?** If YES, what is the project reference number?

NO

4. Explain investigator's and /or supervisor's financial interest, if any, in the outcome of the project.

Nil

5. Do you see any conflict of interest between the interests of the researcher(s), the participants or the funding body? If YES, explain them.

NO

## SECTION K: HUMAN REMAINS, TISSUE AND BODY FLUIDS

1. Are human remains, tissue, or body fluids being used in this research? If YES, complete this section otherwise go to Section L. *The current Human Tissues Act is currently under review and will be changed.*

NO

2. How will the material be taken? For example at operation, urine samples, archaeological digs, autopsy.

3. Is the material being taken at autopsy? YES / NO

If the response to Section J. is YES, provide a copy of the information to be given to the Transplant Coordinator, and state the information that the Transplant Coordinator will provide to those giving consent. Indicate how the material will be stored/disposed of, and explain how the wishes with regard to the disposal of human remains of the whanau (extended family) or similar interested persons will be respected.

4. Is material derived or recovered from archeological excavation? If YES, explain how the wishes of Iwi and Hapū (descent groups), or similar interested persons, or groups, have been respected? YES / NO

5. Will specimens be retained for possible future use? If YES, explain and state this in the PIS. YES / NO

6. Where will the material be stored, and how long will it be stored for?

7. a) Will material remain after the research process? YES / NO

b) How will the material be disposed of? If applicable.

- c) Will material be disposed of in consultation with relevant cultural groups? YES / NO

8. Is blood being collected? YES / NO  
If YES, what volume at each collection, how frequent are the collections, and who is collecting it?

a) Explain how long it will be kept and how it will be stored.

b) Explain how it will be disposed of.

## SECTION L: OTHER MATTERS

1. Have you made any other related applications? If YES, supply approval reference number(s). NO

2. If there is relevant information from past applications or interaction with UAHPEC, please indicate and attach. NO

2. **Are there any other matters you would like to raise that will help the Committee review your application?**

**NO**

---END OF APPLICATION FORM---



**THE UNIVERSITY OF AUCKLAND**  
**NEW ZEALAND**



**Department  
of  
Computer Science**

**Department of Computer Science  
Level 3, Science Centre  
Building 303  
38 Princes St  
Auckland  
Phone 3737599 ext 82128**

## **Appendix B**

### **PARTICIPANT INFORMATION SHEET PARTICIPANT RECRUITMENT**

---

Project title: **A Visual Language and Support Tool for Business Process Modelling**

Researcher name: **Lei Li**

#### **To participants:**

My name is Lei Li. I am a PhD (Computer Science) student at The University of Auckland conducting human computer interaction research. I am conducting this study to assess the usability of a visual language and support tool for business process modelling. I need volunteer participants with a computing or business process modelling background to comment on the usability of my enterprise modelling language prototype system. Up to 30 volunteers will be chosen based on the “first in, first selected” rule.

You are invited to take part in this study and your assistance and comments would be greatly appreciated. The goal of the project is to develop a visual language and its software tool for use in the business process modelling domain that allows the users to build an enterprise level modeling structure. The hypothesis of this research is that the usage of a tree overlay-based visual language significantly reduces the complexity of modelling diagrams and can effectively improve the scalability and collaboration ability with the tool support. I would like to observe you using the prototype software to complete a set of pre-defined tasks, followed by filling in a questionnaire about your experience with the prototype system. I will use the study results to analyse and improve my current PhD research.

The study will take a maximum of one hour. The study can take place at a time convenient to you at the University of Auckland. For industry participants, we may be able to come to your office if required. We are unsure of how usable our notation and tool are for the intended audience - professional business users and business system designers - indeed this is one of the questions that the study hopes to answer. You will not be paid for attending a testing session and any additional expense for travel and food.

You are free to withdraw from the testing session at any time without giving reasons. Data already recorded may not be withdrawn. For the university student participants, choosing either to participate, or not participate in this study will not influence your academic evaluation at the University of Auckland.

A questionnaire will be provided for you to fill in as part of the usability evaluation. The questionnaire will be held in secure storage for six years and then destroyed. The individual questionnaire responses will be summarized and analysed and this summary information may be used both to improve our research outcomes and to report on the findings of the study. A summary of the results of the survey and any resulting publications will be made available to you on request.

All personal information will remain strictly confidential and no material that could personally identify you will be used in any report on this study.

This study has received ethical approval from the University of Auckland Ethics Committee. Please send me a signed Consent Form if you would like to participate in this study. I would like to thank you in advance for your help in making this study possible. Please contact me in either of the following ways if you wish to know more about this study.

**Contact Information:**

Name: Lei Li  
Position: PhD student  
Address: Department of Computer Science, the University of Auckland, Private Bag 92019, Auckland.  
Telephone: 3737599 ext 82128  
Email: [l.li@cs.auckland.ac.nz](mailto:l.li@cs.auckland.ac.nz)

**My supervisors are:**

Professor John Hosking Department of Computer Science  The University of Auckland Private Bag 92019 Auckland <a href="mailto:john@cs.auckland.ac.nz">john@cs.auckland.ac.nz</a>	and	Professor John Grundy Department of Electrical Computer Engineering & Department of Computer Science The University of Auckland Private Bag 92019 Auckland <a href="mailto:john-g@cs.auckland.ac.nz">john-g@cs.auckland.ac.nz</a>
---	-----	---

**The Head of Department is:**

Associate Professor Robert Amor  
Department of Computer Science  
The University of Auckland  
Private Bag 92019  
Auckland  
[trebor@cs.auckland.ac.nz](mailto:trebor@cs.auckland.ac.nz)

**For ethical concerns contact:**

The Chair,  
The University of Auckland Human Participants Ethics Committee, Office of the Vice Chancellor  
Research Office  
Level 2  
76 Symonds Street  
Auckland  
Tel: 373-7599 extn. 87830.

APPROVED BY THE UNIVERSITY OF AUCKLAND HUMAN PARTICIPANTS ETHICS COMMITTEE ON  
...(date)... TO ...(date)...FOR .....(3) YEARS REFERENCE NUMBER 200../...





**UNIVERSITY OF AUCKLAND  
NEW ZEALAND**



**Department  
of  
Computer Science**

**Department of Computer Science  
Level 3, Science Centre  
Building 303  
38 Princes St  
Auckland  
Phone 3737599 ext 82128**

## Appendix C

### CONSENT FORM

*This Consent Form will be held for a period of six years*

---

Project title: **A Visual Language and Support Tool for Business Process Modelling**

Researcher name: **Lei Li**

From Participants:

I have read the Participant Information Sheet, have understood it and I am prepared to take part in the research. I have had the opportunity to ask questions and have them answered. I understand that I am free to withdraw at any time and that data already recorded can not be withdrawn.

- I understand that my responses will be recorded in a questionnaire form.
- I understand that the questionnaire responses may be used to review the research outcomes both to improve the notation and software tool and in publications about the project.
- I understand that I will not be paid for the time taken to participate in this study.
- I understand that data will be archived or stored for six years and then destroyed.

Name \_\_\_\_\_

Date \_\_\_\_\_

Signature \_\_\_\_\_



**UNIVERSITY OF AUCKLAND  
NEW ZEALAND**

UNIVERSITY OF AUCKLAND  
TECHNOLOGY



**Department  
of  
Computer Science**

UNIVERSITY OF AUCKLAND HUMAN PARTICIPANTS ETHICS  
FOR RESEARCH (3) YEARS REFERENCE NUMBER  
**Department of Computer Science  
Level 3 Science Centre  
Building 303  
38 Princes St  
Auckland  
Phone 3737599 ext 82128**

## Appendix D

### PARTICIPANT INFORMATION SHEET USABILITY TESTING

---

Project title: **A Visual Language and Support Tool for Business Process Modelling**

Researcher name: **Lei Li**

To Participants:

My name is Lei Li. I am a PhD student at The University of Auckland conducting research in visual methods to support business process modelling. As a participant of this testing session your feedback will be recorded in response to a number of questions. The questionnaire you are asked to complete will help us gauge the efficiency and effectiveness of our research.

While I would appreciate any assistance you can offer me, your participation is voluntary and will have no effect on your course grade, course participation or commercial benefits in any way.

The questionnaire you are asked to complete is anonymous and none of the information on it will identify you personally. Once completed your questionnaire information cannot be withdrawn. The individual questionnaire responses will be summarised and analysed and this summary information may be used both to improve our research outcomes and to report on the findings of the study. The questionnaire data will be held in secure storage for six years and then destroyed. A summary of the results of the testing and any resulting publications will be made available to you on request.

Please contact me in either of the following ways if you wish to know more about this study.

#### **Contact Information:**

Name: Lei Li

Position: PhD student

Address: Department of Computer Science, the University of Auckland, Private Bag 92019, Auckland.

Telephone: 3737599 ext 82128

Email: [l.li@cs.auckland.ac.nz](mailto:l.li@cs.auckland.ac.nz)

**My supervisors are:**

Professor John Hosking  
Department of Computer Science

and

Professor John Grundy  
Department of Electrical Computer Engineering &  
Department of Computer Science

The University of Auckland  
Private Bag 92019  
Auckland

The University of Auckland  
Private Bag 92019  
Auckland

[john@cs.auckland.ac.nz](mailto:john@cs.auckland.ac.nz)

[john-g@cs.auckland.ac.nz](mailto:john-g@cs.auckland.ac.nz)

**The Head of Department is:**

Associate Professor Robert Amor  
Department of Computer Science  
The University of Auckland  
Private Bag 92019  
Auckland

[trebor@cs.auckland.ac.nz](mailto:trebor@cs.auckland.ac.nz)

**For ethical concerns contact:**

The Chair,  
The University of Auckland Human Participants Ethics Committee, Office of the Vice Chancellor  
Research Office  
Level 2  
76 Symonds Street  
Auckland  
Tel: 373-7599 extn. 87830.

APPROVED BY THE UNIVERSITY OF AUCKLAND HUMAN PARTICIPANTS ETHICS  
COMMITTEE ON ...(date)... TO ...(date)...FOR .....(3) YEARS REFERENCE NUMBER  
200../...

## REFERENCES

---

Ali, N. M. (2007) A Generic Visual Critic Authoring Tool, In Proceedings of the 2007 IEEE Symposium on Visual Languages and Human-Centric Computing, Coeur d'Alène, Idaho, USA.

Anderson P.S. and Apperley M.D. (1990): An interface prototyping system based on Lean Cuisine, *Interacting with Computers Journal*, VOL 2, No 2, 217~226

Apperley M.D. (1988): TaP: A Menu Interface Design Study Using the Lean Cuisine Notation, Information Engineering Report #88/2, Imperial College, London

Baeyens, T. (2007, May 02) The state of workflow,  
<http://www.jbpm.org/state.of.workflow.html>

Barra, E., Génova, G. and Llorens, J. (2004) An approach to aspect modeling with UML 2.0. In Proc. 5th Int. Workshop on Aspect-Oriented Modeling, October 2004.

Baker, J. (2002, June 07) Business Process Modelling Language: Automating Business Relationships, *Business Integration Journal*, [www.bijonline.com](http://www.bijonline.com)

Ballal Rahul, and Michael A. Hoffman (2009) Extending UML for Aspect Oriented Software Modeling. *CSIE*, 488-492

Barrett, D. J., Clarke, L. A., Tarr, P.L., and Wise, A.E. (1996) A Framework for Event-Based Software Integration, *ACM Transactions on Software Engineering and Methodology (TOSEM)*, pp. 378-421.

Bederson, B., Meyer, J. and Good, L. (2000) Jazz: An Extensible Zoomable User Interface Graphics Toolkit in Java, In Proceedings of 2000 ACM Conference on User Interface and Software Technology, ACM Press, pp. 171-180.

Ben-Shaul, I. Z. (1994) Oz: A Decentralized Process Centered Environment. Technical Report CUCS-024-94, Columbia University Department of Computer Science, PhD Thesis.

Benatallah, B., Dumas, M., Fauvet, M.C. and Rabhi, F. (2003) Towards Patterns of Web Services Composition, In Patterns and skeletons for parallel and distributed computing, Springer.

Berndtsson, B., Mellin, J., and Hogberg, U. (1999) Visualization of the Composite Event Detection Process, In proceedings of the 1999 International Workshop on User Interfaces to Data Intensive Systems, IEEE CS Press, pp. 118-127.

Bex Geert Jan, Wim Martens, Frank Neven, and Thomas Schwentick (2005) Expressiveness of XSDs: from practice to theory, there and back again. WWW, 712-721

Bex Geert Jan , Wouter Gelade, Frank Neven, and Stijn Vansummeren (2010) Learning Deterministic Regular Expressions for the Inference of Schemas from XML Data CoRR abs/1004.2372

Box, D., Cabrera, L.F., Critchley, C., Curbera, F., Ferguson, D., Graham, S., Hull, D., Kakivaya, G., Lewis, A., Lovering, B., Niblett, P., Orchard, D., Samdarshi, S., Schlimmer, J., Sedukhin, I., Shewchuk, J., Weerawarana, S., and Wortendyke, D. (2006, January 24) Web Services Eventing (WS-Eventing), <http://www.w3.org/Submission/WS-Eventing/>

BPMI (2010, March 18) <http://www.ebpml.org/bpml.htm>

Buchmann, A., Bornhövd, C., Cilia, M., Fiege, L., Gärtner, F., Liebig, C., Meixner, M., and Mühl, G. (2004) DREAM: Distributed Reliable Event-based Application Management, In Web Dynamics, Springer.

Buschmann F, R. Meunier, and H. Rohnert (1996) Pattern-Oriented Software Architecture. John Wiley and Sons

Budinsky, F., Steinberg, D., Merks, E, Ellersick, R., and Grose, T. (2003) Eclipse Modeling Framework: A Developer's Guide, Addison Wesley Professional.

Burbeck, S. (1992) Applications Programming in Smalltalk-80(TM): How to user Model-View-Controller (MVC), <http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html>

Burnett, M., Atwood, J., Djang, R.W., and Reichwein, J. (2001) Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm, Journal of Functional Programming, 11(2): 155-206.

Burnett, M. and Ambler, A.L. (1992) A Declarative Approach to Event-Handling in Visual Programming Languages, In IEEE Workshop on Visual Languages, pp. 34-40, Seattle, Washington.

Cai, R., Grundy, J.C. and Hosking, J.G. (2007) Synthesizing Client Load Models for Performance Engineering via Web Crawling, IEEE/ACM International Conference on Automated Software Engineering, Atlanta, USA, Nov 5-9 2007, IEEE CS Press.

Chakravarthy, S., Krishnaprasad, V., Anwar, E., and Kim, S.K. (1994) Composite Events for Active Databases: Semantics Contexts and Detection, In Proceedings of the 20th International Conference on Very Large Data Bases.

Chappell, D (2007, April 03) Microsoft and BPM: A Technology Overview, <http://www.microsoft.com/biztalk/solutions/bpm/technicalwhitepaper.msp>

Chen, P. (2002) Entity-relationship modeling Contributions to SE, p296 ~ p310, Springer-Verlag, NY

Cohen, D. (2006, August 15) AP5 Reference Manual, <http://ap5.com/doc/ap5-man.html>

Conway, M., Audia, S., Burnette, T., Cosgrove, D., and Christiansen, K. (2000) Alice: Lessons Learned from Building a 3D System for Novices, In Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 486-493.

Costagliola, G., Deufemia, V., Ferrucci, F., and Gravino, C. (2002) The Use of the GXL Approach for Supporting Visual Language Specification and Interchanging, In Proceedings of HCC'02, Arlington, Virginia, pp.131-138.

Costagliola, G., Deufemia, V. and Polese G. (2004) A Framework For Modeling and Implementing Visual Notations with Applications to Software Engineering, ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 13 Issue 4.

Coupaye, T., Roncancio, C. L., and Bruley, C. (1999) A Visualization Service for Event-Based Systems, Proc. 15emes Journees Bases de Donnees Avancees, BDA.

Cox, P., Giles, F., and Pietrzykowski, T. (1989) Prograph: A step towards liberating programming from textual conditioning, 1989 IEEE Workshop on Visual Languages, Rome, Italy, 150-156.

Cox, P. T., Smedley, T. J., Garden, J. and McManus, M. (1997) Experiences with Visual Programming in a Specific Domain – Visual Language Challenge '96, In Proceedings of the 1997 IEEE Symposium on Visual Languages, pp. 254-259.

Cugola G., Di Nitto E., and Fuggetta A. (1998) Exploiting an event-based infrastructure to develop complex distributed systems, In Proceedings of the 20th international conference on Software engineering, Kyoto, Japan, pp. 261-270.

Dantas Daniel S., David Walker, Geoffrey Washburn, and Stephanie Weirich (2008) AspectML: A polymorphic aspect-oriented functional programming language. ACM Trans. Program. Lang. Syst. (TOPLAS) 30(3)

Dehnert Juliane (2003) Making EPCs fir for Workflow Management. EMISA Forum (EMISA) 23(1), 12-26

Dewan, P. and Choudhary, R. (1991) Flexible user interface coupling in collaborative systems, CHI'91, pp. 41-49.

Dillon, A (2001), Usability evaluation, In W. Karwowski (ed.) Encyclopedia of Human Factors and Ergonomics, London: Taylor and Francis.

Dogac Asuman, Yusuf Tambag, Pinar Pembecioglu, Sait Pektas, Gokce Laleci, Gokhan Kurt, Serkan Toprak, and Yildiray Kabak (2002) An ebXML infrastructure implementation through UDDI registries and RosettaNet PIPs. SIGMOD, 512-523

Draheim D and G. Weber (2005) Form-Oriented Analysis, Springer-Verlag Berlin Heidelberg

Drumea, A. and Popescu, C. (2004) Finite State Machines and Their Applications in Software for Industry Control, Electronics Technology: Meeting the Challenges of Electronics Technology Progress, 27th International Spring Seminar, Vol.1, pp. 25-29.

ebPML (2002, June 06) BPML 1.0 Analysis,  
[http://www.ebpml.org/bpml\\_1\\_0\\_june\\_02.htm](http://www.ebpml.org/bpml_1_0_june_02.htm)

Eclipse (2009, August 21) EMF, <http://www.eclipse.org/modeling/emf/>

Eclipse (2009, August 21) GEF, <http://www.eclipse.org/gef/>

Embarcadero (2010) <http://www.embarcadero.com/products/er-studio>

Effinger Philip and Johannes Spielmann (2010) Lifting business process diagrams to 2.5 dimensions. VDA, 75300

Engels, G. and Erwig M. (2005) ClassSheets: Automatic Generation of Spreadsheet Applications from Object-Oriented Specifications, In Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, Long Beach, CA, USA, pp. 124-133.



- Eriksson, H.E. and Penker, M., (2000). Business modeling with UML: business patterns at work, Wiley
- Felfernig, A., Friedrich, G., Jannach, D., Russ, C. and Zanker M. (2003) Developing Constraint-Based Applications with Spreadsheets. In 18th International Joint Conference on Artificial Intelligence. Acapulco, Mexico.
- Feng Yi-Heng and Lee C. Joseph (2010) Exploring Development of Service-Oriented Architecture for Next Generation Emergency Management System. AINA Workshops, 557-561
- Fensel, D. and Bussler, C. (2002) The web service modeling framework WSMF, Electronic Commerce Research and Applications, vol. 1, no. 2, pp. 113--137.
- Fernstrom, C. (1993) Process Weaver: Adding Process Support to UNIX. In Proceedings of the Second International Conference of Software Process. Pages 12-26.
- Ferguson R., Parrington N., Dunne P., Archibald J. and Thompson J. (1999) MetaMOOSE-an object-oriented framework for the construction of CASE tools, In Proceedings of CoSET'99, LA.
- Foster, H., Uchitel, S., Magee, J. and Kramer, J. (2003) Model-based verification of web service compositions, In Proceedings of the 18th IEEE international conference on automated software engineering, Montreal, Canada.
- Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995) Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley.
- Gansner Emden R., Yehuda Koren and Stephen C. North (2004) Topological Fisheye Views for Visualizing Large Graphs. INFOVIS, 175-182
- Gatzui, S and Dittrich, K.R. (1993) Events in an Active Object-Oriented Database System, Proc. 1st Intl. Workshop on Rules in Database Systems (RIDS).

Goel A (2006) Enterprise Integration --- EAI vs. SOA vs. ESB, Infosys Technologies White Paper

Gokhale, A. and Gray J. (2005) An Integrated AOMD Development Toolsuite for Distributed Real-Time and Embedded Systems, Proc 6th I W- AOM, Chicago

Gottfried, H. J. and Burnett M. M. (1997) Programming Complex Objects in Spreadsheets: an Empirical Study Comparing Textual Formula Entry with Direct Manipulation and Gestures, Papers Presented at the Seventh Workshop on Empirical Studies of Programmers ESP'97.

Green, T. R. G. and Petre, M. (1996) Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *J. Visual Languages and Computing*, 7, 131-174.

Grosse Philippe, Yves Durand and Paul Feautrier (2009) Methods for power optimization in SOC-based data flow systems. *ACM Trans. Design Autom. Electr. Syst. (TODAES)* 14(3)

Grundy, J.C. and Hosking, J.G. (1995) ViTABaL: A Visual Language Supporting Design by Tool Abstraction, Proceedings of the 1995 IEEE Symposium on Visual Languages, Darmstadt, Germany, IEEE CS Press, pp. 53-60.

Grundy, J.C. and Hosking, J.G. (1996) Constructing Integrated Software Development Environments with MViews. *International Journal of Applied Software Technology*, vol. 2, no. 3-4, 133-160.

Grundy, J.C. and Hosking, J.G. (1998) Serendipity: integrated environment support for process modelling, enactment and work coordination, *Automated Software Engineering: Special Issue on Process Technology* 5(1), January 1998, Kluwer Academic Publishers, pp. 27-60.

Grundy, J.C., Hosking, J.G., Li, L. and Liu, N. (2006) Performance engineering of service compositions, ICSE 2006 Workshop on Service-oriented Software Engineering, Shanghai, China.

Grundy, J.C., Hosking, J.G., and Mugridge, W.B. (1996) Supporting flexible consistency management via discrete change description propagation, *Software – Practice and Experience*, Volume 26, Issue 9, pp. 1053 – 1083.

Grundy, J.C., Hosking, J.G. and Mugridge, W.B. (1996) Towards a unified event-based software architecture, in *Joint Proceedings of the SIGSOFT'96 Workshops, 1996 International Software Architecture Workshop, Oct 14-15, San Francisco*, ACM Press, 121-125.

Grundy, J. C., Hosking, J. G. and Mugridge, W. B. (1997) Visualising Event-based Software Systems: Issues and Experiences. In *Proceedings of SoftVis97*. Adelaide, Australia.

Grundy, J.C., Hosking, J.G., Zhu N., and Liu N. (2006) Generating Domain-Specific Visual Language Editors from High-level Tool Specifications, In *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, Tokyo, Japan, pp. 25-36.

Grundy, J.C., Mugridge, W.B. and Hosking, J.G. (1998a) Visual specification of multiple view visual environments, In *Proceedings of IEEE VL'98*, Halifax, Nova Scotia, Canada, IEEE CS Press, pp. 236-243.

Grundy, J.C., Mugridge, W.B., and Hosking, J.G. (1998b) Static and Dynamic Visualisation of Software Architectures for Component-based Systems, In *Proceedings of the 10th International Conference on Software Engineering and Knowledge Engineering*, San Francisco, KSI Press, pp. 426-433.

Gugola G, Nitto E, and Fuggetta A. (2001) The JEDI event-based infrastructure and its application to the development of the OPSS WFMS, *IEEE Trans. Software*, 2001, 27(9): 827-850.

Haeberli, P. (1988) ConMan: a visual programming language for interactive graphics, In Proceedings of the 15th annual conference on Computer graphics and interactive techniques, ACM Press, pp. 103-111.

Hamadi, R. and Benatallah, B. (2003) A petri-net based model for web service composition, Proc 14th Australasian Database Conference, Adelaide, Australia, CRPIT Press.

Hanna, K. (2002) Interactive visual functional programmings, Proceedings of the seventh ACM SIGPLAN international conference on Functional programming ICFP '02, Volume 37 Issue 9.

Hanson, J. (2005, September 01) Event-driven services in SOA, Java World, <http://www.javaworld.com/javaworld/jw-01-2005/jw-0131-soa.html>

Hartson, H. R., Andre, T. S., and Williges, R. C. (2003) Criteria for evaluating usability evaluation methods, International Journal of Human-Computer Interaction 15(1): 145-181.

Haskell (2007, December 02), <http://www.haskell.org/>

Hill, R. D. (1992) The Abstraction-Link-View Paradigm: Using Constraints to Connect User Interfaces to Applications. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Monterey, California, pp.335-342, ACM Press.

Hartmann Sven, Sebastian Link and Thu Trinh (2009) Constraint acquisition for Entity-Relationship models. Data Knowl. Eng. (DKE) 68(10):1128-1155

Hill, R. D., Brinck, T., Rohall, S. L., Patterson, J. F. and Wilner, W. (1994) The Rendezvous Architecture and Language for Constructing Multiuser Applications. ACM Transactions on Computer-Human Interaction (TOCHI). Vol. 1, no. 2, pp.81-125, ACM Press.

Hirakawa, M. and Ichikawa, T. (1992) Advances in Visual Programming, In Proceedings of the Second International Conference on Systems Integration, pp. 538-543.

Hoof, J.V. (2006, July 27) how EDA extends SOA and why it is important, <http://soa-eda.blogspot.com/2006/11/how-eda-extends-soa-and-why-it-is.html>

Hudak, P. (1989) Conception, evolution, and application of functional programming languages, ACM Computing Surveys 21 (3): 359-411.

Huh, J., Grundy, J.C., Hosking, J.G., Liu, N. and Amor R. (2007) Integrated data mapping for meta-tool model integration, transformation and code generation, working paper, the University of Auckland.

IBM (2010, March 21) Business Processes Web Services for Java, <http://www.alphaworks.ibm.com/tech/bpws4j>

IBM (2009, April 09) Specification: Business Process Execution Language for Web Services Version 1.1, <http://www.ibm.com/developerworks/library/ws-bpel/>

Inazumi, H. and Omoto, N. (1999) A new scheme for verifying rule-based systems using Petri nets, 1999 IEEE International Conference on Systems, Man, and Cybernetics, Volume 1, Page(s):860 - 865 vol.1.

Information Week (2009, May 21): [www.informationweek.com](http://www.informationweek.com)

Jacob, R. (1996) A Visual Language for Non-WIMP User Interfaces, In Proceedings of the 1996 IEEE Symposium on Visual Languages, Boulder, CO, USA, pp. 231-238.

Jamroendararasame K., Suzuki T., and Tokuda T. (2003) A visual approach to development of Web services providers/requestors. In the 2003 IEEE Symposium on Visual and Multimedia Software Engineering, pages 251--253

Jia, X., Steele, A., Qin L., Liu, H. and Jones, C. (2005) An Event-Based Framework for Model Integration, In Proceedings of the 2005 IEEE International Conference on Electro Information Technology.

Jin, W. (2003) A structured approach to visualising the event handling specification, Thesis (MSc-Computer Science) – University of Auckland.

Jong E.D. (1997) Multi-paradigm Programming in Large Control Systems, In Proceedings of the 1997 Joint Workshop on Parallel and Distributed Real-Time Systems (WPDRTS/OORTS '97).

JUDE (2010, July 04): <http://jude.change-vision.com/jude-web/product/professional.html>

Jung, M.C. and Cho, S.B. (2005). A novel method based on behavior network for Web service composition, In Proceedings of the International Conference on Next Generation Web Services Practices, 22-26 Aug. 2005. Page(s):6 pp.

Kelly, S., Lyytinen, K., and Rossi, M. (1996) Meta Edit+: A Fully configurable Multi-User and Multi-Tool CASE Environment, In Proceedings of CAiSE'96, LNCS 1080.

Kelso, J. (2002) A Visual Programming Environment for Functional Languages, Thesis (PhD – Computer Science) – Murdoch University, <http://www.csse.uwa.edu.au/~joel/vfpe/thesis.pdf>

Kienzle Jörg, Wisam Al Abed and Jacques Klein (2009) Aspect-oriented multi-view modeling. AOSD, 87-98

Kiringa, I. (2002) Specifying Event Logics for Active Database, Description Logics

Koenig, J. (2004, June 04) JBOSS jBPM, [http://www.jboss.com/pdf/jbpm\\_whitepaper.pdf](http://www.jboss.com/pdf/jbpm_whitepaper.pdf)

Kornkamol J., S. Tetsuya, and T. Takehiro, (2003) "A Visual Approach to Development of Web Services Providers/Requestors", Proc. of VL/HCC'03, Auckland, p251~p253

Kraemer, F. A. and Herrmann P. (2007) Transforming Collaborative Service Specifications into Efficiently Executable State Machines, In Proceedings of the Sixth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2007).

Krishnamurthy, B. and Rosenblum, D.S. (1995) Yeast: a general purpose event-action system, In Proceedings of Software Engineering, IEEE Transactions, pp. 845-857.

Ledeczi A., Bakay A., Maroti M., Volgyesi P., Nordstrom G., Sprinkle J., and Karsai G. (2001) Composing Domain-Specific Design Environments, Computer, 44-51.

Lewicki, D. and Fisher, G. (2006) VisiTile - A Visual Language Development Toolkit, Proceedings of the 1996 IEEE Symposium on Visual Languages, Boulder, Colorado, pp. 114-121.

Li, K.N.L., Hosking, J.G., Grundy, J.C. and Li, L. (2009): 'Visualising Event-based Information Models: Issues and Experiences', Visual Analytics in Software Engineering, Workshop at 2009 IEEE/ACM Automated Software Engineering Conference, Proceedings of Visual Analytics in Software Engineering, Auckland, New Zealand, 16 Nov, 2009

Li L., Hosking J.G., and Grundy J.C (2008): MaramaEML: An Integrated Multi-View Business Process Modelling Environment with Tree-Overlays, Zoomable Interfaces and Code Generation, In Proceedings of the 23th IEEE/ACM International Conference on Automatic Software Engineering (ASE 08), L'Aquila, Italy (Best Software Demo Award)

Li L., Hosking J.G., and Grundy J.C (2007): Visual Modelling of Complex Business Processes with Trees, Overlays and Distortion-Based Displays, In Proceedings of the 2007 IEEE Conference on Visual Languages/Human-Centric Computing (VL HCC 07), Coeur d'Alène, Idaho, U.S.A

Li, L., Grundy, J.C. and Hosking, J.G. (2007) EML: A Tree Overlay-based Visual Language for Business Process Modelling, In Proceedings of the 2007 International Conference on Enterprise Information Systems, Portugal.

Li, L., Phillips, C.H.E. and Scogings C.J., (2004) Automatic Generation of a Graphical Dialogue Model from Delphi, Proc of APCHI 2004, Rotorua, p221~p230

Li, X., Marin, J. M. and Chapa, S. V (2002) A Structural Model of ECA Rules in Active Database, Lecture Notes in Computer Science, in Proceedings of the Second Mexican International Conference on Artificial Intelligence, Pages 486-493

Li, X., Mugridge W. B. and Hosking G. (1997) A Petri Net-based Visual Language for Specifying GUIs, In Proceedings of the 1997 IEEE Symposium on Visual Languages, Isle of Capri, Italy.

Liu, A.F., Chen Z.G.; He H. and Gui W.H. (2007) Treenet: A Web Services Composition Model Based on Spanning tree, In Proceedings of the 2nd International Conference on Pervasive Computing and Applications, 26-27 July 2007 Page(s):618 – 623

Liu, N., Hosking, J.G. and Grundy, J.C. (2005) A Visual Language and Environment for Specifying Design Tool Event Handling, In Proc. VL/HCC'2005, Dallas, Texas, USA.

Liu, N., Grundy, J.C. and Hosking, J.G. (2007) A Visual Language and Environment for Specifying User Interface Event Handling in Design Tools, In Proceedings of the 2007 Australasian Conference on User Interfaces, Ballarat, Australia, CRPIT Press.



Liu, N., Hosking, J.G. and Grundy, J.C. (2004) Integrating a Zoomable User Interfaces Concept into a Visual Language Meta-tool Environment, In Proceedings of the 2004 International Conference on Visual Languages and Human-Centric Computing, Rome, Italy, IEEE CS Press.

Liu, N., Hosking, J.G. and Grundy, J.C. (2007) MaramaTatau: Extending a Domain Specific Visual Language Meta Tool with a Declarative Constraint Mechanism, in Proceedings of the 2007 IEEE Symposium on Visual Languages and Human-Centric Computing, Coeur d'Alène, Idaho, USA

Liu, N., Grundy, J.C. and Hosking, J.G. (2005) A visual language and environment for composing web services, In Proceedings of the 2005 ACM/IEEE International Conference on Automated Software Engineering, Long Beach, California, IEEE Press, pp. 321-324.

Myers B.A. (1995 a): State of the Art in User interface Software Tools, Reading in Human-Computer interaction: Toward the Year 2000, Morgan Kaufmann Publishers Inc, 323~343

Myers B.A. (1995 b): User Interface Software Tools, ACM Transactions on Computer-Human Interaction. Vol. 2, no. 1, March 1995. 64 ~ 103

Matskin, M. and Montesi, D. (1998) Visual Rule Language for Active Database Modelling, Information Modelling and Knowledge Bases IX. IOS Press, pp. 160-175.

Marshall C. (2004) Enterprise Modelling with UML. Designing Successful Software Through Business Analysis, Addison Wesley

Martinez A and M. Patino (2005) “ZenFlow: A Visual Web Service Composition Tool for BPEL4WS”, Proc of VL/HCC'05, Dallas, P181~P188

Meier, R. and Cahill, V. (2002) Taxonomy of Distributed Event-based Programming Systems, In Proceedings of the 22nd International Conference on Distributed Computing Systems Workshop.

Menon, S., Dasgupta, P., and LeBlanc, R.J. (1993) Asynchronous event handling in distributed object-based systems. In Proc. the 13th Conference on Distributed Computing Systems, pages 383-390, Pittsburgh, Pennsylvania.

MIT (2006, October 12) The Alloy Analyzer, <http://alloy.mit.edu/>

Morch, A. (1998) Tailoring tools for system development, Journal of End User Computing, 10:2, pp. 22-29.

Moreira Orlando, Twan Basten, Marc Geilen and Sander Stuijk (2010) Buffer Sizing for Rate-Optimal Single-Rate Data-Flow Scheduling Revisited. IEEE Trans. Computers (TC) 59(2):188-201

MSDN (2007, November 14) Create Trigger, [http://msdn2.microsoft.com/en-us/library/aa258254\(SQL.80\).aspx](http://msdn2.microsoft.com/en-us/library/aa258254(SQL.80).aspx)

MSDN (2007, November 14) Understanding the Event Model, <http://msdn2.microsoft.com/en-us/library/ms533023.aspx>

MSDN (2005, February 21) DSL Tools, <http://msdn2.microsoft.com/en-us/vstudio/aa718368.aspx>

Mugridge, W.B., Hosking, J.G. and Grundy, J.C. (1998) Drag-throughs and attachment regions in BuildByWire, Proc. of OZCHI'98, Adelaide, Australia, IEEE CS Press, pp. 320-327.

Myers, B. (1990) A. Garnet: Comprehensive Support for Graphical, highly Interactive User Interfaces. IEEE COMPUTER. 23 (11), 71-85.

- Myers, B.A. (1997) The Amulet Environment: New Models for Effective User Interface Software Development, IEEE TSE, vol. 23, no. 6, 347-365.
- Myers, B.A., Pane, J.F., and Ko, A. (2004) Natural programming languages and environments, <http://www.acmqueue.org/>
- Narayanan, S. and McIlraith, S.A. (2002) Simulation, verification and automated composition of web services. In Proceedings of the 11th World Wide Web Conference.
- Neag, I. A. and Tyler, D. F. (2001) Combined Visual and Textual Programming Methodology for Signal-based Automatic Avionics Testing Systems, In Proceedings of the 20th Conference on Digital Avionics System, 9A5/1-9A5/10 vol.2.
- Nikau (2007, July 08) Marama, <http://www.cs.auckland.ac.nz/Nikau/marama/>
- OMG (2009, April 25) BPMN 2.0, <http://www.bpmn.org/>
- OMG (2004, September 17) Event Service Specification, <http://www.omg.org/docs/formal/04-10-02.pdf>
- OMG (2003, October 07) OCL, <http://www.omg.org/docs/ptc/03-10-14.pdf>
- Palanque, P.A., Bastide, R., Dourte, L. and Sibertin-Blanc, C. (1993) Design of User-Driven Interfaces Using Petri Nets and Objects, In Proceedings of Advanced Information Systems Engineering, pp. 569-585, Springer-Verlag.
- Paschke, A. (2006) ECA-LP / ECA-RuleML: A Homogeneous Event-Condition-Action Logic Programming Language, Int. Conf. on Rules and Rule Markup Languages for the Semantic Web (RuleML'06), Athens, Georgia, USA.
- Pautasso, C. and Alonso, G. (2003) Visual Composition of Web Services, Proc IEEE HCC'03, Auckland, pp. 92-99.

Pautasso, C. and Alonso, G. (2005) The JOpera Visual Composition Language JVLC, 16(1-2):119-152.

Peltonen J. (2000) Visual Scripting for UML-Based Tools. In Proceedings of ICSSEA 2000: Paris, France.

Phillips C.H.E (1992): Extending Lean Cuisine: A Case Study in Reverse Engineering, School of Information Science, MASSEY University

Phillips C.H.E. (1993): Software Support for Lean Cuisine+, PhD Thesis, MASSEY University, 157~171

Phillips C.H.E. (1994a): Review of Graphical Notations for Specifying Direct Manipulation Interfaces. *Interacting with Computers*, 411~431

Phillips C.H.E. (1994b): Serving Lean Cuisine+: Towards a Support Environment, Proceedings OZCHI94, CHISIG of Ergonomics Soc. of Australia, 41~46

Phillips C.H.E (1995): Lean Cuisine+: An Executable Graphical Notation for Describing Direct Manipulation Interfaces. *Interacting with Computers*, 49~71

Phillips C.H.E and Apperley M.D (1990): Direct Manipulation Interaction Tasks: A Macintosh-Based Analysis, School of Information Science, MASSEY University

Phillips C.H.E and Kemp E. (2001): Extending UML use case modeling to support graphical user interface design, Proceedings of ASWEC 2001, IEEE, Canberra, Australia, 48 ~ 57

Phillips C.H.E and Kemp E. (2002): In Support of User Interface Design in the Rational Unified Process, Proceedings of the Third Australasian User Interface Conference, Australian Computer Society, 21~27

Phillips C.H.E and McKaige J. (1997): OZCHI'96 Industry Session, Sixth Australian Conference on Human-Computer Interaction, Department of Computer Science, University of Waikato

Phillips C.H.E and Scogings C. (1997): Modelling the mock-up: towards the automatic specification of the behaviour of early prototypes, Interact '1997, IFIP, Chapman and Hall, London, 591~592., Sydney, Australia

Phillips C.H.E and Scogings C. (1998a): Task and Dialogue Modelling: Bridging the Divide with Lean Cuisine+, Proceedings of the First Australasian User Interface Conference, IEEE, 81~87

Phillips C.H.E and Scogings C. (1998b): Towards the automatic specification of the behavior of early prototypes using Lean Cuisine+, Proc of Software Engineering: Education and Practice (SE:E&P'98), IEEE, Dunedin, January 1998, 238~244

Plaisant, C. and Shneiderman, B. (2005) Show me! Guidelines for producing recorded demonstrations, in Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing, Dallas, USA, 171-178.

Rapide Design Team (1997) Program Analysis and Verification Group, Computer Systems Lab, Stanford University, Guide to the Rapide 1.0 Language Reference Manuals.

Recker, J. (2010a): Continued Use of Process Modeling Grammars: The Impact of Individual Difference Factors. European Journal of Information Systems, Vol. 19, No. 1, pp. 76-92.

Recker, J. and Rosemann, M. (2010): The Measurement of Perceived Ontological Deficiencies of Conceptual Modeling Grammars. Data & Knowledge Engineering, Vol. 69, No. 5, pp. 516-532.

Recker, J. (2010b): Opportunities and Constraints: The Current Struggle with BPMN. Business Process Management Journal, Vol. 16, No. 1, pp. 181-201.

- Recker, J. and Rosemann, M. (2009): Teaching Business Process Modeling: Experiences and Recommendations. *Communications of the Association for Information Systems*, Vol. 25, No. 32, pp. 379-394.
- Recker, J., Rosemann, M., Indulska, M. and Green, P. (2009): Business Process Modeling: A Comparative Analysis. *Journal of the Association for Information Systems*, Vol. 10, No. 4, pp. 333-363.
- Recker, J., Young, R., Darroch, F., Marshall, P. and McKay, J. (2009): ACIS 2007 Panel Report: Lack of Relevance in IS Research. *Communications of the Association for Information Systems*, Vol. 24, No. 18, pp. 303-314.
- Recker, J. and Niehaves, B. (2008): Epistemological Perspectives on Ontology-based Theories for Conceptual Modeling. *Applied Ontology*, Vol. 3, No. 1-2, pp. 111-130.
- Recker, J (2008): Understanding Process Modelling Grammar Continuance, PhD thesis, Queensland University of Technology
- Recker, J., Rosemann, M. and Krogstie, J. (2007): Ontology- versus Pattern-based Evaluation of Process Modeling Languages: A Comparison. *Communications of the Association for Information Systems*, Vol. 20, No. 48, pp. 774-799.
- Recker, J. (2007): A Socio-Pragmatic Constructionist Framework for Understanding Quality in Process Modelling. *Australasian Journal of Information Systems*, Vol. 14, No. 2, pp. 43-63.
- Recker, J. and Indulska, M. (2007): An Ontology-Based Evaluation of Process Modeling with Petri Nets. *Journal of Interoperability in Business Information Systems*, Vol. 2, No. 1, pp. 45-64.
- Repenning, A. and Sumnet, T. (1995) Agentsheets: a medium for creating domain-oriented visual languages, *Computer*, 28, no. 3.

Robbins, J. E., Medvidovic, N., Redmiles, D. F. and Rosenblum, D. S. (1998) Integrating Architecture Description Languages with a Standard Design Method, In Proceedings of the 20th International Conference on Software Engineering, Kyoto, Japan, pp. 209-218.

Robbins, J.E. and Redmiles D.F. (1998) Software architecture critics in the Argo design environment, J Knowledge-Based Systems 11 (1998) 47-60.

Roberts, D, and Johnson, R. (1996) Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks, In Pattern Languages of Program Design 3, Addison-Wesley, Reading, MA.

RuleML Initiative (2006, May 20), <http://www.ruleml.org>

Schnieders, A. and Puhlmann, F. (2005) Activity Diagram Inheritance. In Proc of the 8th ICBIS, Poland, p3 ~ p15

Schiffer, S. and Fröhlich, J.H. (1994) Concepts and Architecture of Vista – a Multiparadigm Programming Environment, in Proceedings of the 10th IEEE/CS Symposium on Visual Languages, St.Lois/USA, pp. 40-47.

Sheth, B.D. (1994) A Learning Approach to Personalized Information Filtering, Master thesis in Computer Science and Engineering, Massachusetts Institute of Technology.

Shin Kitae, Chankwon Park, HyounGon Lee and Jinwoo Park (2005) Efficient Mapping Rule of IDEF for UMM Application. ICCSA, 1219-1228

Singh Darryl, Mitra Nataraj and Rick Mugridge (2004) ViewPoint: A Zoomable User Interface for Integrating Expressive Systems. APCHI, 631-635

Sliwa, C. (2003, December 10), Event-driven architecture poised for wide adoption, Computer World, <http://www.computerworld.com/softwaretopics/software/appdev/story/>

Smith, D.C., Cypher, A. and Spohrer, J. (1995) KidSim: programming agents without a programming language, *Communications of the ACM*, vol. 37, no. 7, pp. 54 – 67.

Smith, D. N. (1990) The interface construction set, in *Visual Languages and Applications*, (T. Ichikawa, E. Jungert, R. Korfhage, eds.), Plenum Pub., NY.

Spönemann Miro , Hauke Fuhrmann, Reinhard von Hanxleden and Petra Mutzel (2009) Port Constraints in Hierarchical Layout of Data Flow Diagrams. *Graph Drawing*, 135-146

Srivastava, B. and Koehler, J. (2003) Web Service Composition - Current Solutions and Open Problems, ICAPS Workshop on Planning for Web Services, Trento, Italy.

Sun (2005, December 02) Java BluePrints Model-View-Controller,  
<http://java.sun.com/blueprints/patterns/MVC-detailed.html>

Sun (2007, November 30) The J2EETM Tutorial,  
<http://java.sun.com/javaee/5/docs/tutorial/doc/>

Sun (2007, November 30) The Java™ Tutorials, Lesson: Writing Event Listeners,  
<http://java.sun.com/docs/books/tutorial/uiswing/events/index.html>

Suzuki Masami (1992) A Method of Utilizing Domain and Language specific Constraints in Dialogue Translation. *COLING*, 756-762

Taentzer, G. (1999) Adding Visual Rules to Object-Oriented Modeling Techniques, in *Proceedings of Technology of Object-Oriented Languages and Systems (TOOLS'99)*, Nancy, France. IEEE Computer Society.

Tang, Y., Chen, L. He, K.T. and Jing, N. (2004). SRN: an extended Petri-net-based workflow model for Web service composition, In *Proceedings of the IEEE International Conference on Web Services*, 6-9 July 2004. Page(s):591 – 599



Thalheim Bernhard (2009) Extended Entity-Relationship Model. Encyclopedia of Database Systems, 1083-1091

Thone, S., Depke, R. and Engels, G. (2002) Process-oriented, flexible composition of web services with UML, Proc ER-Wkshp on Conceptual Modeling Approaches for e-Business, Tampere, Finland, LNCS, 2002

Tigris (2010, April 09) <http://argouml.tigris.org/>

Tolvanen J. (2006) OOPSLA demonstrations chair's welcome: MetaEdit+: integrated modeling and metamodeling environment for domain-specific languages, Companion to the 21st ACM.

Uchitel Sebastián, Robert Chatley, Jeff Kramer and Jeff Magee (2003): LTSA-MS: Tool Support for Behaviour Model Elaboration Using Implied Scenarios. TACAS, 597-601

Urbas. L., Nekarsova. L. and Leuchter. S. (2005) State chart visualization of the control flow within an ACT-R/PM user model, In Proc. IWTMD05, p43~p48.

Vlissides, J.M. and Linton, M. (1989) Unidraw: A framework for building domain-specific graphical editors, Proc. UIST'89, ACM Press, pp. 158-167.

W3C (2001, May 26), Web Services Description Language (WSDL) 1.1, <http://www.w3.org/tr/wsdl>

Wagner, F., Schmuki R., Wagner T. and Wolstenholme P. (2006) Modeling Software with Finite State Machines: A Practical Approach, Auerbach Publications.

Weber, G. (2003) Semantics of form-oriented analysis, <http://www.diss.fu-berlin.de/2003/72/>

Welch, B. and Jones, K. (2003) Practical Programming in Tcl and Tk, Prentice-Hall.

Wirtz, G. (1993) A Visual Approach for Developing, Understanding and Analyzing Parallel Programs, in Proc IEEE VL'93, IEEE CS Press, pp. 261-266.

Wordsworth, J.B. (1992) Software Development with Z - A Practical Approach to Formal Methods in Software Engineering, Addison Wesley.

Workflow Management Coalition (1999, October 20) Terminology & Glossary, [http://www.wfmc.org/standards/docs/TC-1011\\_term\\_glossary\\_v3.pdf](http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf)

Zapletal Marco, Wil M. P. van der Aalst, Nick Russell, Philipp Liegl and Hannes Werthner (2009) An Analysis of Windows Workflow's Control-Flow Expressiveness. ECOWS 200-209

Zhu, N., Grundy, J.C., Hosking, J.G., Liu, N., Cao, S. and Mehra, A. (2007) Pounamu: a meta-tool for exploratory domain-specific visual language tool development, Journal of Systems and Software 80 (8), Elsevier.