

Title	Hazard and early warning analysis based on domain specific modeling technologies
Authors	Imran, Syed
Publication date	2013
Original Citation	Imran, S., 2013. Hazard and early warning analysis based on domain specific modeling technologies . PhD Thesis, University College Cork.
Type of publication	Doctoral thesis
Rights	© 2013, Syed Imran - http://creativecommons.org/licenses/by-nc-nd/3.0/
Download date	2024-04-19 09:20:50
Item downloaded from	https://hdl.handle.net/10468/1038

HAZARD AND EARLY WARNING ANALYSIS
BASED ON DOMAIN SPECIFIC MODELING
TECHNOLOGIES

SYED IMRAN

M.Sc. COMPUTER SCIENCE

TECHNISCHE UNIVERSITÄT BRAUNSCHWEIG, DEUTSCHLAND



A THESIS SUBMITTED TO THE NATIONAL UNIVERSITY OF IRELAND, CORK
IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY IN THE FACULTY OF SCIENCE

October 2012

Dr Ioannis Dokas Supervisor

Dr Ian Pitt Co-Supervisor

Professor Barry O'Sullivan Head of Department

Department of Computer Science,
National University of Ireland, Cork.

Executive Summary

An aim of proactive risk management strategies is the timely identification of safety related risks. One way to achieve this is by deploying early warning systems. Early warning systems aim to provide useful information on the presence of potential threats to the system, the level of vulnerability of a system, or both of these, in a timely manner. This information can then be used to take proactive safety measures. The United Nation's has recommended that any early warning system need to have four essential elements, which are the risk knowledge element, a monitoring and warning service, dissemination and communication and a response capability. This research deals with the risk knowledge element of an early warning system.

The risk knowledge element of an early warning system contains models of possible accident scenarios. These accident scenarios are created by using hazard analysis techniques, which are categorised as traditional and contemporary. The assumption in traditional hazard analysis techniques is that accidents are occurred due to a sequence of events, whereas, the assumption of contemporary hazard analysis techniques is that safety is an emergent property of complex systems.

The problem is that there is no availability of a software editor which can be used by analysts to create models of accident scenarios based on contemporary hazard analysis techniques and generate computer code that represent the models at the same time.

This research aims to enhance the process of generating computer code based on graphical models that associate early warning signs and causal factors to a hazard, based on contemporary hazard analyses techniques. For this purpose, the thesis investigates the use of Domain Specific Modeling (DSM) technologies.

The contributions of this thesis is the design and development of a set of three graphical Domain Specific Modeling languages (DSML)s, that when combined together, provide all of the necessary constructs that will enable

safety experts and practitioners to conduct hazard and early warning analysis based on a contemporary hazard analysis approach. The languages represent those elements and relations necessary to define accident scenarios and their associated early warning signs. The three DSMLs were incorporated into a prototype software editor that enables safety scientists and practitioners to create and edit hazard and early warning analysis models in a usable manner and as a result to generate executable code automatically.

This research proves that the DSM technologies can be used to develop a set of three DSMLs which can allow user to conduct hazard and early warning analysis in more usable manner. Furthermore, the three DSMLs and their dedicated editor, which are presented in this thesis, may provide a significant enhancement to the process of creating the risk knowledge element of computer based early warning systems.

Acknowledgements

First of all, I would like to thank my supervisor Dr Ioannis Dokas for his guidance, encouragement and advice that he provided to me throughout this research which ensured the successful completion of this thesis. I found his mentoring and coaching approach to be most invaluable, where he used lectures and workshops to effectively explain complex technical concepts and ensure that I had a clear understanding of them. His research knowledge and expertise has inspired me to undertake further research.

I would like to acknowledge the moral support and prayers of my mother Marium and my younger brother Syed Haider. Undoubtedly, I would not have been able to complete this research without their unconditional love and care. Also, a constant motivation for me to successfully complete this research was my father's philosophy in life, to realise our full potential and serve humankind.

I am grateful to all current and previous members of the Cork Constraint Computation Centre (4C), especially my colleagues - Franclin Foping, Walid Trabelsi, Abdul Razak and Lisa Swenson; the administrative staff - Eleanor O'Riordan, Linda O'Sullivan, Catriona Walsh and Olivia Coleman and the technical staff Peter MacHale and Joe Scanlon. I would like to express my appreciation to John Feehan who I have enjoyed worked closely with to surmount the challenges of this research and for his help in this research.

I wish to thanks my friends Asif Imtiaz Ranjha, Syed Azhar Shah, Sajjad Khilji and in particular Mirza Safdar Baig for all the emotional support, camaraderie, entertainment, and caring they provided.

I would like to express my sincere appreciation to the following people who assisted me with my research, including Dr Ian Pitt, University College Cork; Dr Richard Wallace, University College Cork; Dr Bartel Van de Walle, Tilburg University; Dr Dimitrios Kolovos, York University; Dr Igor Kozine, Technical University of Denmark; Mr Brendan Goggin, and the staff of Cork City Council; and Mr Pat Murphy, and the staff of Cork County Council.

Finally, I would like to acknowledge the support of the Irish Environmental Protection Agency for this research project SCEWA (Grant No 2007-DRP-2-S5) under the DERP grant scheme.

Dedication

TO MY FATHER, SYED MAHDI WHO WOULD HAVE BEEN PROUD TO SEE THIS
THESIS COMPLETED.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.

Signed:
Syed Imran

Contents

Executive Summary	ii
Acknowledgements	iv
Dedication	vi
Declaration	vii
1 Introduction	1
1.1 Background	1
1.2 Motivation and Problem Statement	2
1.3 Aim and Research Objectives	5
1.4 Hypothesis	8
1.5 Research Contribution	8
1.6 Dissertation Outline	9
2 Safety and Early Warning Sign Analysis	11
2.1 Accident Models	11
2.1.1 STAMP	13
2.2 STPA	15
2.3 Justifying Perceivable Signs as Early Warnings	17
2.4 EWaSAP Steps	19
2.5 Hazard and Early Warning Analysis	20
2.6 Summary	21
3 Methods and Techniques for Domain-Specific Modeling	23
3.1 Model Driven Development	23
3.2 Strategies for Defining DSMLs	27
3.3 Defining DSMLs From Scratch	29
3.3.1 Abstract Syntax	30
3.3.1.1 Domain Analysis	30

3.3.1.2	Meta-Model	31
3.3.2	Concrete Syntax	32
3.4	Model Transformation	33
3.5	Meta-modeling Tools that Support the Creation of DSMLs . . .	34
3.5.1	Generic Modeling Environment	34
3.5.2	MetaEdit+	35
3.5.3	Microsoft DSL Tools	36
3.5.4	Eclipse GMF	36
3.6	Summary	37
4	A Review of DSMLs for Early Warning Systems	38
4.1	Examples of DSMLs Application Domains	38
4.2	DSMLs With Hazard Analysis Concepts	39
4.2.1	SOPHIA	39
4.2.2	EAST-ADL	42
4.3	DSML for Risk Knowledge Modeling of Early Warning Systems	43
4.3.1	The Stream-Oriented DSML	43
4.3.2	The OpenCOM and Transition Diagrams DSML	45
4.4	Discussion and Conclusions	49
5	Software Technologies Used for the Development of the Hazard and Early Warning Analysis DSML	51
5.1	The Development Platform	51
5.2	The Main Components of the GMF Framework	52
5.2.1	Eclipse Modeling Framework	53
5.2.2	Graphical Editing Framework	53
5.2.3	EVL	54
5.2.4	EGL	55
5.2.5	XML, Java and XPath	55
5.2.6	PostgreSQL and JDBC	56
5.3	Summary	56
6	Hazard and Early Warning Analysis DSML: Requirements, Specifica- tions and Architecture	57
6.1	Defining the Meta-models	57
6.1.1	Hazard and Early Warning Analysis Task Identification	58
6.1.2	Hazard and Early Warning Analysis Concepts and Se- mantics	59
6.1.3	DSML Meta-model Requirements	61

6.2	Meta-Model Specifications	64
6.3	Concrete Syntax Specifications	70
6.3.1	Selection of Graphical Icons	70
6.3.2	Concrete Syntax Specifications	72
6.3.3	Validations	73
6.4	DSML Architecture	76
6.5	Summary	78
7	Design and Architecture of the Hazard and Early Warning Analysis Editor	79
7.1	Prototype Software Editor Requirements	79
7.2	Architecture	81
7.2.1	Dedicated Editors Layer	82
7.2.2	Ecore Model Layer	86
7.2.3	The Code Generation Layer	86
7.2.4	The Database Layer	89
7.3	Workflow	90
7.4	Summary	94
8	Evaluation	95
8.1	Overview of the Evaluation Process	95
8.2	Methods	96
8.2.1	Usability Evaluation Methods	96
8.2.2	Benchmarking Methods	98
8.3	Evaluation Approach	99
8.3.1	The Reference System	100
8.3.2	The Participants Groups	102
8.4	1st Evaluation Phase	102
8.4.1	Indicative Example	104
8.4.2	SUS Data Collection and Analysis	111
8.4.3	Discussion of the 1st Evaluation Phase	120
8.5	2nd Evaluation Phase	121
8.5.1	Benchmarking Data and Analysis	123
8.6	Summary	127
9	Summary and Conclusion	129
9.1	Summary of the Thesis	129
9.2	Contribution of this Research	131
9.3	Critical Remarks	132
9.3.1	On the Potential Usefulness of the Research	133

9.4	Future Research Directions	133
9.4.1	Web Based Accessibility of the DSMLs	134
9.4.2	Creating Hazard and Early Warning Analysis Models in a Collaborative Manner	134
9.4.3	Compare the Updating Process of the Risk Knowledge Element of a Real Early Warning System	135
9.4.4	Reasoning Support	135
9.5	Concluding Remarks	136
	Glossary	137
	Appendices	139
	A Questionnaires	140
	References	165

List of Tables

2.1	The sequence of hazard and early warning analysis steps . . .	21
6.1	The main tasks and subtasks of the hazard and early warning analysis after hazard identification step.	58
6.2	Domain concepts associated with first main task and their meanings.	59
6.3	Domain concepts associated with second main task and their meanings	60
6.4	Domain concepts associated with third main task and their meanings	61
6.5	Requirements necessary for the execution of at least two main tasks of the domain	62
6.6	Requirements necessary for the execution of a first main task of the analysis	62
6.7	Requirements necessary for the execution of a second main task of the analysis	63
6.8	Requirements necessary for the execution of a third main task of the analysis	64
6.9	Ecore concepts and their graphical representation in GMF . .	65
6.10	Selection of graphical icons for the first main task	71
6.11	Selection of graphical icons for the second main task	71
6.12	Selection of graphical icons for the third main task	72
6.13	Validations over user models	73
6.14	Enforced model validations on the domain concepts of the first main task	73
6.15	Enforced model validations on the domain concepts of the second main task	74
6.16	Enforced model validations on the domain concepts of the third main task	75

7.1	Functional requirements hazard of the prototype editor	80
7.2	Usability requirements of the hazard and early warning analysis prototype editor.	81
8.1	Calculation of the SUS score based on the User 01 ratings of the questionnaire shown in Figure 8.16.	114
8.2	Calculation of the SUS score based on the User 02 ratings of the questionnaire.	114
8.3	Calculation of the SUS score based on the User 03 ratings of the questionnaire.	115
8.4	Calculation of the SUS score based on the User 04 ratings of the questionnaire.	115
8.5	Calculation of the SUS score based on the User 05 ratings of the questionnaire.	116
8.6	Calculation of the SUS score based on the User 06 ratings of the questionnaire.	116
8.7	Calculation of the SUS score based on the User 01 ratings of the questionnaire shown in Figure 8.17 - Effectiveness Aspect.	117
8.8	Calculation of the SUS score based on the User 02 ratings of the questionnaire - Effectiveness Aspect.	117
8.9	Calculation of the SUS score based on the User 03 ratings of the questionnaire - Effectiveness Aspect.	118
8.10	Calculation of the SUS score based on the User 04 ratings of the questionnaire - Effectiveness Aspect.	118
8.11	Calculation of the SUS score based on the User 05 ratings of the questionnaire - Effectiveness Aspect.	119
8.12	Calculation of the SUS score based on the User 06 ratings of the questionnaire - Effectiveness Aspect.	119
8.13	The overall SUS scores of each expert for the satisfaction aspect of the prototype editor.	120
8.14	The overall SUS scores of each expert for the effectiveness of the DSMLs constructs.	120
8.15	Time of each expert to complete the main tasks of the analysis with and without prototype software editor.	123
8.16	Data analysis of the times needed to complete the first main task of the analysis.	124
8.17	Data analysis of the times needed to complete the second main task of the analysis.	124

8.18 Data analysis of the times needed to complete the third main task of the analysis.	125
8.19 The time difference between manual analysis and with the use of the prototype editor.	126

List of Figures

1.1	Common approach for translating the risk knowledge into code for an early warning system.	3
1.2	An improved approach for translating the risk knowledge into code for an early warning system.	4
1.3	The context of this research and its boundaries of the proposed solution.	7
2.1	The domino model of accident causation (Qureshi, 2008). . . .	12
2.2	Feedback control process.	14
2.3	A feedback control process with generic control process flaws.	17
2.4	Early warning sign justification model.	18
2.5	A more detailed view of the early warning signal justification model.	19
3.1	The four layer MDD architecture (Atkinson and Kuhne, 2003).	25
3.2	The model transformation layer as part of the generic MDD architecture.	27
3.3	SysML taxonomy diagram (Boutekkouk et al., 2009).	28
3.4	Ecore kernel (Budinsky and Brodsky, 2003).	32
4.1	The refinement of the UML meta-model with the fundamental concepts of SOPHIA.	39
4.2	A SOPHIA model with elements from the "Accident" package (Cancila et al., 2009a).	40
4.3	Sophia and SysML integration model.	41
4.4	Snippet of meta-model specifications for EAST-ADL.	43
4.5	Meta-model specification of Stream-Oriented DSML (Sadilek, 2007b).	44
4.6	A user defined earthquake detection algorithm model (Sadilek, 2007a).	44
4.7	OpenCom meta-model (Bencomo, 2008).	46

4.8	The meta-model to model the context and environment variability (Bencomo, 2008).	47
4.9	Overview of the approach implemented by Genie (Bencomo, Grace, Flores, Hughes and Blair, 2008a).	48
5.1	The three layers of eclipse architecture (Gamma and Beck, 2004).	52
5.2	The GEF model view controller pattern (Hudson and Shah, 2005).	54
6.1	Ecore meta-model specifications.	66
6.2	DSML meta-model specific for the first main task.	67
6.3	DSML meta-model specific for the second main task.	68
6.4	DSML meta-model specific for the third main task.	69
6.5	Snippet of annotations specified for meta-model of ICAML.	72
6.6	Snippet of EVL code.	75
6.7	Validation on user model.	76
6.8	The architecture of the proposed solution comprised by three DSMLs.	77
7.1	The consecutive tasks and subtasks of the domain and the three DSMLs.	80
7.2	The architecture of the prototype hazard and early warning analysis editor.	82
7.3	Generation of a GMF based editor from the domain model.	83
7.4	Graphical icon to generate code on the menu bar.	84
7.5	Snippet of the java program used to insert an "id" value.	85
7.6	Database schema with its referential integrity constraints	85
7.7	Snippet of EGL template for generating Java code.	87
7.8	Generated Java code.	88
7.9	Snippet of EGL template for generating XML code.	88
7.10	Generated XML code.	89
7.11	Snippet of the java program used to store the generated code into database.	89
7.12	UML Activity diagram of prototype early warning sign analysis DSML software tool.	90
7.13	The CSML editor.	91
7.14	The ICAML editor.	92
7.15	The AML editor.	93
7.16	The Database Explorer editor.	94
8.1	Part of the chlorination system of water treatment works.	101
8.2	The concept of "Hazard" specified using AML editor.	104

8.3	Defining a "Control structure diagram" using the CSML editor.	105
8.4	The property view of a component in a "Control structure diagram" using the CSML editor.	105
8.5	Assigning "Inadequate control actions" using the ICAML editor.	106
8.6	The "Inadequate control action" dialogue in ICAML editor. . .	106
8.7	Defining the "Process model" used to control water chlorination process.	107
8.8	Updating the "Process model" used to control water chlorination process.	107
8.9	Assigning control process flaws using the AML editor.	108
8.10	Assigning of possible generic control process flaw type.	108
8.11	Specifying "Early warning signs" using AML editor.	109
8.12	Attributes of "Early warning sign".	109
8.13	Snippet of XML code generated using AML editor.	110
8.14	Snippet of Java code generated using AML editor.	110
8.15	A Query performed using database explorer editor.	111
8.16	User satisfaction ratings over the satisfaction aspect of the usability of the prototype editor.	112
8.17	User satisfaction ratings over the effectiveness of the DSMLs constructs.	113
8.18	Average time to complete the main tasks of the domain.	126
A.1	User 01 ratings of the SUS questionnaire on satisfaction aspect of prototype software editor.	141
A.2	User 01 ratings of the SUS questionnaire satisfaction aspect of prototype software editor (Page 2).	142
A.3	User 01 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs.	143
A.4	User 01 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs (Page 2).	144
A.5	User 02 ratings of the SUS questionnaire on satisfaction aspect of prototype software editor.	145
A.6	User 02 ratings of the SUS questionnaire on satisfaction aspect of prototype software editor (Page 2).	146
A.7	User 02 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs.	147

A.8	User 02 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs (Page 2).	148
A.9	User 03 ratings of the SUS questionnaire on satisfaction aspect of prototype software editor.	149
A.10	User 03 ratings of the SUS questionnaire on satisfaction aspect of prototype software editor (Page 2).	150
A.11	User 03 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs.	151
A.12	User 03 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs (Page 2).	152
A.13	User 04 ratings of the SUS questionnaire on satisfaction aspect of prototype software editor.	153
A.14	User 04 ratings of the SUS questionnaire on satisfaction aspect of prototype software editor (Page 2).	154
A.15	User 04 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs.	155
A.16	User 04 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs (Page 2).	156
A.17	User 05 ratings of the SUS questionnaire on satisfaction aspect of prototype software editor.	157
A.18	User 05 ratings of the SUS questionnaire on satisfaction aspect of prototype software editor (Page 2).	158
A.19	User 05 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs.	159
A.20	User 05 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs (Page 2).	160
A.21	User 06 ratings of the SUS questionnaire on satisfaction aspect of prototype software editor.	161
A.22	User 06 ratings of the SUS questionnaire on satisfaction aspect of prototype software editor (Page 2).	162
A.23	User 06 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs.	163
A.24	User 06 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs (Page 2).	164

Introduction

1.1 Background

A proactive approach to risk management aims to prevent possible accidents or reduce their effects if they happen. According to Rasmussen, a proactive approach aims at designing a strategy based on the following points (Rasmussen and Svedung, 2000):

- i) Identify the boundaries of safe performance;
- ii) Make the boundaries of safe performance visible to decision makers;
- iii) Counteract pressures that drive decision-makers toward these boundaries.

An important phase of a proactive risk management strategy is the identification of safety related risks. Early warning systems are tools that provide useful and timely information on the presence of potential threats and vulnerabilities in a system. Thus, early warning systems can make visible to decision makers the position where the state of a system lies in relation to the boundaries of its safe performance. They are considered in many cases as important tools within the context of proactive risk management.

Early warning systems are used in a wide spectrum of domains. These domains can be considered to be members of a wider taxonomy of systems like, for example:

- a) Natural and environmental systems, focusing on natural hazards (Zschau and Küppers, 2003), such as earthquakes (Erdik et al., 2003), landslides (Zan et al., 2002), floods (Artan et al., 2002), volcanic threats (Ewert et al., 2005), tsunami (Rudloff et al., 2009), and other natural hazards.

- b) Socio-technical systems, where accidents can occur in utilities and critical infrastructures such as in drinking water treatment plants (Brosnan, 1999), and landfills (Dokas, Karras and Panagiotakopoulos, 2009).
- c) Systems where their purpose is to control political emergencies, such as conflicts (Zenko and Friedman, 2011), humanitarian crises (WHO, 2006) and genocides (Woocher, 2006).
- d) Financial systems, like those associated with currency and banking crises (Bussiere and Fratzscher, 2006) (Tung et al., 2004).

Variations in the characteristics of different domains is one reason for the non-existence of a universally acceptable definition of an early warning system (Dokas, Wallace, Marinescu, Imran and Foping, 2009). Nonetheless, the United Nations have recommended that in order to be complete and effective, an early warning system needs to have the following four elements (UN/ISDR, 2006):

- i) Risk knowledge: Collected with appropriate risk assessment and hazard analyses approaches.
- ii) Monitoring and a warning service: To provide accurate and timely forecasting of hazards, using reliable scientific methods and technologies that enable monitoring of the appropriate hazards and vulnerabilities.
- iii) Dissemination and communication services: To disperse understandable and meaningful warnings so that it can be useful to those at risk.
- iv) Response capability: In order to prepare communities to respond and act on detectable threats.

1.2 Motivation and Problem Statement

The motive of this research is to enhance the creation of the risk knowledge elements of early warning systems dealing with critical infrastructures safety issues. The risk knowledge element in such early warning systems contains models of possible accident scenarios. These accident scenarios are created by using hazard analysis techniques. The hazard analysis techniques guide analysts to identify the contributing factors to a loss and to assess the risks involved within a domain.

Hazard analysis techniques can be categorised as traditional and contemporary. When traditional hazard analysis techniques, such as fault tree

analysis and event tree analysis are used, the analysts adopt the assumption that accidents occur due to a sequence of events, similar to the behaviour of domino blocks. When one block falls, it causes the others to fall as well. Conversely, when the contemporary hazard analysis techniques are adopted the analyst accepts that safety is a dynamic control problem (Rasmussen and Svedung, 2000), (Stringfellow et al., 2010), (Dulac and Leveson, 2004) and, as such, they must identify and mitigate the factors which contribute to the enforcement of inadequate control actions.

The different perspective, which the assumptions of the contemporary hazard analysis techniques enforces to the analysts, enables them to identify a more complete set of contributing factors to a loss, such as organisational, cultural and software factors (Leveson, 2012), compared to the traditional techniques.

In practice, when developing computer based early warning systems, an analyst should first select the appropriate hazard analysis technique and then create models of possible accident scenarios. He/she needs then to transform the hazard analysis models into software code, to form the risk knowledge element of the early warning system, as depicted in Figure 1.1.

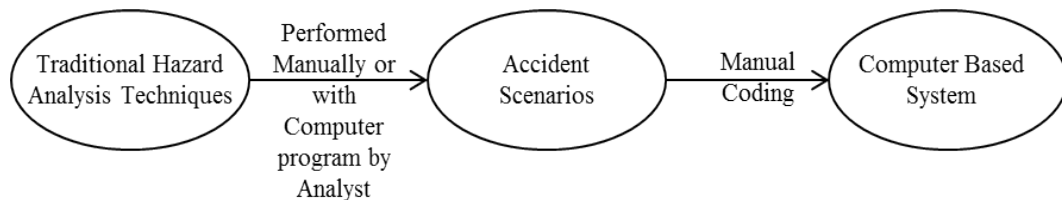


Figure 1.1: Common approach for translating the risk knowledge into code for an early warning system.

The issues associated with the process of accident scenario creation are the following:

- a) Typically, the analysts responsible for conducting risk analysis choose traditional hazard analysis techniques. One contributing factor for not using contemporary techniques is the fact that these have been introduced a few years ago, and hence, have not been fully comprehended and adopted by the majority of analysts.
- b) A significant set of early warning signs and signals may not be recognised during the analysis, regardless of the type of hazard analysis technique used. The problem in the case where a traditional hazard analysis is used,

as mentioned above, is that these are not capable to help analysts define a number of contributing factors to a loss, making the identification of early warning signs for these causes an impossible task. Whereas, in the case where a contemporary hazard analysis technique is used, the problem is the lack of steps for the identification of early warning signs for each contributing factors to a loss which has been identified. This issue has been recognised by the systems safety engineering community. It is stated, for example, by Professor Woods when he concluded in one of his articles, that *"the safety field still lacks the concepts, tools, and measures to recognise warning signs prior to major failure events"* (Woods, 2009).

Vulnerabilities of the process shown in Figure 1.1 include:

- a) Possible mistakes in coding, resulting in flaws and bugs in the risk knowledge element of an early warning system;
- b) Time delays in the manual coding process;
- c) The additional time and resources, which are needed in order to adequately update the models of accident scenarios and to generate the necessary computer code.

One way of addressing these issues and vulnerabilities is by providing to the analyst a computer program that would allow him/her, as shown in Figure 1.2, to:

- a) Create, edit and update in an easy and usable manner what is referred in this dissertation as "hazard and early warning analysis" models, which are models of a contemporary hazard analysis technique that have been extended with the necessary steps to enable analysts in defining the relations between the early warning signs and losses, via their corresponding causal factors;
- b) Generate executable code based on these models without any manual coding.

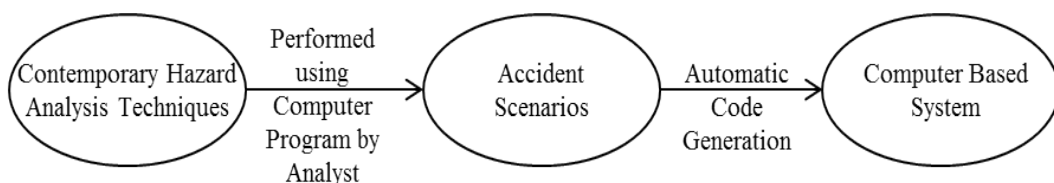


Figure 1.2: An improved approach for translating the risk knowledge into code for an early warning system.

There are software editors which help analysts create hazard analysis models. Characteristic examples are PLFaultCAT (Dehlinger and Lutz, 2006), CASEHAT (Yet-Pole, 2003) and ReliaSoft's Xfmea (Cooperation, 2003).

PLFaultCAT (Product-Line Fault Tree Creation and Analysis Tool) is a software safety analysis tool with interactive graphical user interface for applying Software Fault Tree Analysis (SFTA) technique to investigate contributing causes to potential hazards in safety-critical applications. It is an extension of the FaultCAT application (Burgess, 2004), which is an open-source traditional fault tree creation tool written in Java programming language.

CASEHAT (Computer Aided Semiconductor Equipment Hazard Analysis Tool) is a computer based hazard analysis tool, which is used to carry out mainly hazard and operability analysis (HAZOP) for semiconductor manufacturing processes. It was developed using Microsoft Access with the support of Microsoft Visual Basic for Applications (VBA) programming language.

The Xfmea software tool (Cooperation, 2003) facilitates to conduct Failure Modes and Effects Analysis (FMEA), and Failure Modes, Effects and Criticality Analysis (FMECA). It is based on a spreadsheet format with basic user interface with customisation options to fit your particular analysis.

However, these editors can create traditional hazard analysis models only and, furthermore, these editors do not have the capability of generating automatically executable code based on the models which have been created.

Thus, there is a need to create a new software tool that will allow analysts to:

- a) Create models of accident scenarios based on a contemporarily hazard analysis;
- b) Relate early warning signs with losses;
- c) Generate executable code based on these models.

1.3 Aim and Research Objectives

The aim of this research is *to enhance the creation of risk knowledge models together with their associated early warnings and their transformation into computer code, so as to help analysts in the creation process of early warning systems for critical infrastructures*. The main objective is *to create a new graphical modeling language, based on the fundamental concepts of a contemporary hazard analysis technique, which will be incorporated into a prototype software editor using Domain Specific Modeling (DSM) technologies*.

DSM is a software development methodology that raises the level of abstraction beyond programming by specifying solutions directly using domain concepts (Tolvanen, 2004). It is a methodology, where highly specific modeling languages are used in order to generate software via fully automatic code generation directly from the models (Ansorg and Schwabe, 2010).

With DSM technologies graphical computer languages can be designed and developed. Each graphical element of the language represents a model of a concept of the problem domain and it can be connected with other graphical elements based on a predefined syntax. With models, the focus is drawn on the important aspects of the problem domain. A domain is an area of knowledge or activity characterised by a set of concepts and terminology that is understood by the experts in the domain (Booch et al., 1999).

Domain specific graphical modeling languages differ from general purpose programming languages such as JAVA, Python or domain specific programming languages such as Structured Query Language (SQL), eXtensible Markup Language (XML) and Hyper Text Markup Language (HTML), which require programming skills. The main skill required for using graphical DSM languages is an adequate knowledge of the problem domain. Furthermore, software tools developed based on DSM technologies are capable of producing computer code in programming languages, thanks to code generator mechanisms. These mechanisms are responsible for transforming the graphical models into a programming language code.

The domain in this dissertation is a contemporary hazard analysis technique, which has recently been extended with additional steps that enable analysts to create relations between the early warning signs with losses via their corresponding causal factors. Specifically, at the core of the domain is the STAMP-Based Hazard Analysis (STPA), which is a contemporary hazard analysis technique based on systems theory that was introduced by Professor Leveson (Leveson, 2004a), (Leveson, 2002a), (Leveson, 2002b). STPA have been used for the design of safety critical systems and has proved that it is more comprehensive compared to traditional hazard analysis techniques (Leveson, 2004a), (Ishimatsu et al., 2010), (Stringfellow, 2011) (Leveson, 2012).

The extension of the STPA, is the Early Warning Sign Analysis Using STPA (EWaSAP) (Dokas et al., 2012), which provides additional steps, to help analysts identify one or many early warning signs for each contributing factor to a loss. The result of the integration of EWaSAP into STPA is what will be referred into this dissertation as a "hazard and early warning analysis" approach.

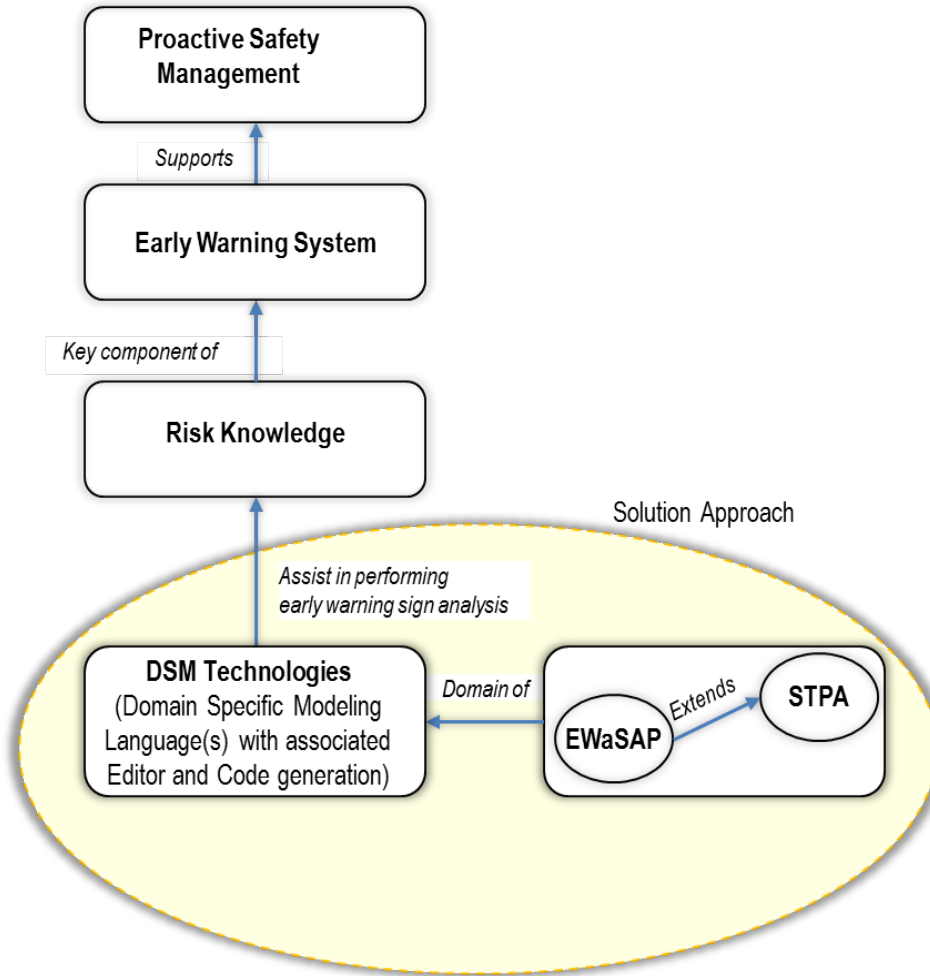


Figure 1.3: The context of this research and its boundaries of the proposed solution.

Pursuing a way of achieving a solution to the problem, similar to the one shown in Figure 1.3 and described above, implies dealing with the following objectives:

- a) Identify the proper development strategies of the DSM graphical languages that will allow the execution of the tasks of the domain. In particular, for the design and development of a graphical DSM, the available alternatives were to refine the concepts of Unified Modeling Language (UML) using the UML profiling mechanism and to design and develop it from scratch using proper open source technologies. Both alternatives were studied and evaluated in this dissertation with respect to the aspects of the domain which should be represented as well as to their code generation capabilities.
- b) Select the DSM technologies that will be used to develop the prototype software editor. Indeed, there are a number of software technologies that

needs to be integrated, to create an editor of a graphical DSM with code generation capabilities. A description of these technologies and how they were used to create a novel prototype hazard and early warning analysis editor will be presented in this dissertation.

- c) Assess the effectiveness of the graphical Domain Specific Modeling Language (DSML)s and the usability of its prototype software editor. The results of a usability evaluation, as well as of a benchmarking that measures the time analysts take to conduct the analysis with the use of the prototype editor, compared to a manual application of the analysis, will be shown.

In essence, the research questions raised in this thesis are the following:

1. *Which DSM technologies would be helpful to develop a prototype software editor that can be used by an analyst to perform hazard and early warning analysis? How can these technologies be combined to create an effective hazard and early warning analysis tool with code generation support?*
2. *How many graphical DSMs are needed to cover all aspects of the problem domain? How many constructs should each language have? How will the constructs of the DSML be associated and interact with each other?*

1.4 Hypothesis

The hypothesis made in this research is that *the graphical DSMLs enables experts to perform hazard and early warning analysis in a usable manner*. According to International Organization for Standardization (ISO) Standard 9241-Part 11 (ISO, 1998), usability expresses the extent to which a product can be used by specified users to achieve certain goals with effectiveness, efficiency and satisfaction in a specified context of use. In order to support this hypothesis the prototype software editor of the solution approach was evaluated by a set of users (i.e. the human user of the language who is expert in the particular domain) for its effectiveness, efficiency and satisfaction after creating a number of hazard and early warning analysis models.

1.5 Research Contribution

The contribution claimed by this dissertation is a set of three graphical DSMLs that when combined together, provide all of the necessary constructs that

will enable safety experts and practitioners to conduct hazard and early warning analysis. The elements that comprise the DSMLs represent those elements and relations necessary to define accident scenarios and their associated early warning signs. The three DSMLs were incorporated in to a prototype software editor that enables safety scientists and practitioners to create and edit hazard and early warning analysis models in a usable manner and as a result to generate executable code automatically. Therefore, this contribution may provide a significant enhancement to the process of creating the risk knowledge element of computer based early warning systems.

1.6 Dissertation Outline

The rest of the thesis is structured as follows:

Chapter 2, *Safety and Early Warning Sign Analysis*, introduces fundamental concepts of the hazard and early warning analysis, which is the domain of the DSMLs to be designed and developed in this research.

Chapter 3, *Methods and Techniques for Domain-Specific Modeling*, presents the basic concepts and necessary techniques for adopting the DSM approach. The chapter also describes the alternative strategies for developing DSML.

Chapter 4, *A Review of DSMLs for Early Warning Systems*, surveys previous work and reports those DSMLs that were capable of creating domain specific hazard analysis models and that were developed for the creation of early warning systems.

Chapter 5, *Software Technologies Used for the Development of the Early Warning Sign Analysis DSML*, describes open source software technologies and modeling frameworks selected for the design and development of the hazard and early warning analysis DSMLs.

Chapter 6, *Hazard and Early Warning Analysis DSML: Requirements, Specifications and Architecture*, describes the main contribution of this research, which are the requirements and specifications of the hazard and early warning analysis DSMLs, along with their architecture.

Chapter 7, *Design and Architecture of the Hazard and Early Warning Analysis*

Editor, focuses on the design and architecture of a prototype software editor, which was created to enable users to conduct hazard and early warning analysis from the use of the DSMLs, and generate software code from the models of their analysis.

Chapter 8, *Evaluation*, presents a usability evaluation that assesses the effectiveness, satisfaction and effectiveness on the use of hazard and early warning analysis DSMLs via their prototype software editor.

Finally, in Chapter 9, *Summary and Conclusion*, presents the summary of each chapter followed by the research contribution. The chapter concludes the thesis by providing critical remarks, and future research directions, along with the concluding remarks.

Safety and Early Warning Sign Analysis

THIS chapter introduces key concepts that were used to build up the constructs of the hazard and early warning analysis DSMLs. Emphasis is given to the Systems Theoretic Accident Models and Processes (STAMP), which is a contemporary model that explains the phenomenon of accidents, and to the STPA, which is a contemporary hazard analysis approach that conforms with the assumptions of STAMP. Furthermore, the EWa-SAP early warning sign analysis is explained, since it is used as an extension to STPA to help analysts identify the early warning signs for each contributing factor to a loss.

2.1 Accident Models

An accident is defined as an undesired and unplanned (but not necessarily unexpected) event that results in (at least) a specified level of loss (Leveson, 1995). Safety is the freedom from accidents. In order to explain the phenomenon of accidents, scientists and practitioners are making use of accident models. The accident models provide descriptions of the assumptions and of the elementary components that are needed to explain the phenomenon of accidents. According to Hollnagel (Hollnagel, 2004), accident models can be classified as

- (a) sequential;
- (b) epidemiological;
- (c) systemic.

The sequential models explain accidents causation as the result of clearly distinguishable events that occur in a specific order and imply that the way to prevent the accident is to break the sequential order. A representative example of the sequential models is Heinrich's domino theory (Heinrich, 1931) as shown in Figure 2.1. Domino theory states that accidents result from a chain of sequential events, where one event triggers the other like a line of dominoes falling over. In order to avoid an accident it is necessary to remove a key factor (i.e. a domino from the line).

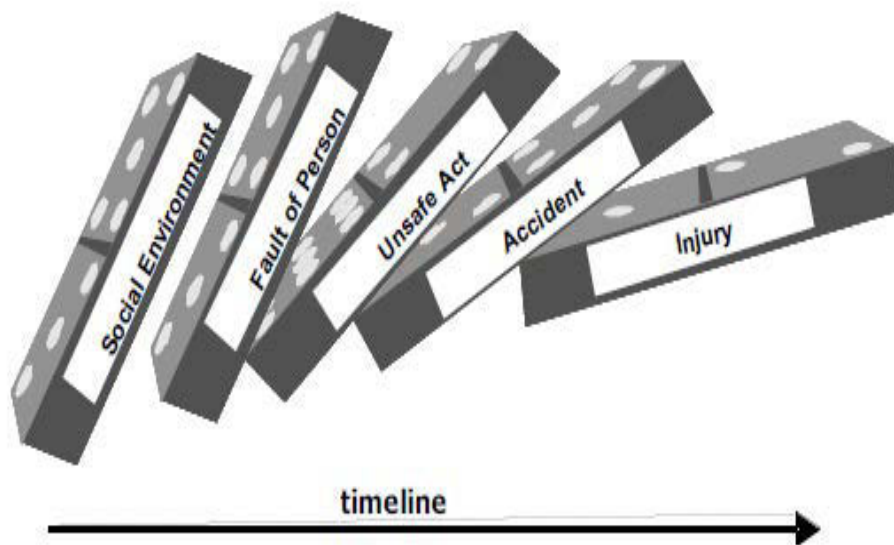


Figure 2.1: The domino model of accident causation (Qureshi, 2008).

The epidemiological models explain accidents with a set of factors, some of which are obvious and some are latent. Characteristic example of epidemiological model is Reason's Swiss cheese model (Reason and Reason, 1997). According to this model an accident occurs when hazards (i.e. a harmful agents, large amount of energy, etc.) can pass through the successive layers of defenses of the system and reach the vulnerable people or assets. An accident thus entail the breaching of barriers, either these are hard (i.e. technical devises, automated safety features, interlocks) or soft (i.e. legislations, permit works etc.). The "erosion" of defenses is attributed to:

- a) Active failures, such as errors and violations at the sharp end of the system by workers;
- b) Latent conditions, such as gaps in supervision, poor design, shortfall in training that may lie dormant for years within the system, similar to a pathogen of the human body, and become apparent only when are combined with active failures resulting to accidents.

A contemporary explanation of the phenomenon of accidents is provided by the systemic accident models. According to these models accidents result due to dysfunctional and in some cases unexpected interactions between system components, which can affect the performance of the entire system. Examples of systemic models include the Functional Resonance Accident Model (FRAM) (Hollnagel, 2004); the Traffic Organisation and Perturbation AnalyZer (TOPAZ) (Blom et al., 2001); and the STAMP (Leveson, 2004b).

2.1.1 STAMP

STAMP is a new accident model based on systems theory that considers safety as a control problem (Rasmussen and Svedung, 2000), (Dulac and Leveson, 2004). According to systems theory, systems can be modeled as hierarchical structures where each level imposes laws that restrict or allows certain behaviour in the lower level. STAMP considers two basic hierarchical control structures, one for system development and one for system operation with interactions between them (Leveson, 2004c). For example, a system development control structure at the top level of the hierarchy may include government departments, which define the purpose and the acceptable behaviours of the elements of the socio-technical system under study via government policies.

State agencies may be located in the immediate lower level. They are authorised by the government departments to provide operational permits, to conduct audits and to assess the performance of the elements in the lower hierarchical levels. Furthermore, they report any findings back to the government departments.

The management of the companies who design and develop the socio-technical system of the study may be located in the hierarchical levels that are directly below state agencies. Each company during the design phase contains its own hierarchical structure, which may include the project management with its design and development levels and the manufacturing management with its different levels of manufacturing processes. During the operational phase, in addition to the management and manufacturing levels, the companies contain the operational management level with its different levels of operational processes.

Among the different hierarchical levels are safety-related control laws or constraints that specify what constitute the non-hazardous system states. All safety constraints are enforced with control processes. Even the physical design of the system aiming at controlling any harmful condition such as the

release of energy results from a set of control processes during the system design phase. Control processes can be described with a model similar to that shown in Figure 2.2, which is a typical *feedback control process*.

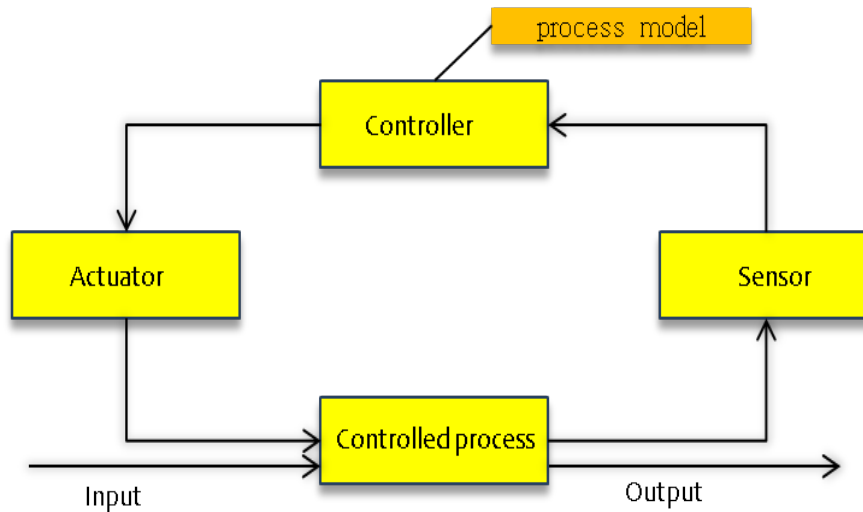


Figure 2.2: Feedback control process.

The model consists of the *sensor*, *controller* and *actuator* elements whose combined objective is to control the behaviour of a *controlled process*. The sensor element interacts with the controlled process and generates streams of data providing thus feedback information to the controller. The controller interprets what is being perceived by the sensor and tries to comprehend if the controlled process is in a safe state or not. During this task the controller "classifies" some of the received data as early warning signs. This happens because the controller is the element of the feedback control process that possesses the process models and the accident scenarios that give meaning to the stream of perceived data and puts them in context. The controller then decides, based on these models, if it is necessary to apply changes into the controlled process via actions executed by the actuator. Therefore, the term *early warning sign* in this dissertation implies the value of an observation, or of a series of observations, made by the sensor element, which according to the mental models possessed by the controller indicates the presence of causal factors to a potential loss.

The feedback control processes provide, among other things, the information necessary to impose safety laws on the lower hierarchical levels and measurements indicating how effectively the safety constraints were enforced. Any controller, human or automated, in a feedback control process must

contain a model of the system being controlled. For humans, this model is generally referred to as their **mental model** of the process being controlled and for the automated systems is generally referred to as their **process model**.

Accidents may happen because the models used by the controller to describe the behaviour of the process being controlled can become inconsistent to the actual state of the controlled process. Hence, it is necessary to understand and eliminate the contributing factors that may result in any inconsistencies between the model of the process used by the controllers and the actual state of the controlled process.

Systems in STAMP are viewed as interrelated components that are kept in a state of dynamic equilibrium by feedback control processes of information and control (Leveson et al., 2006) and because of that the **causal factors**, of an accident are not limited only to a series of events. Accidents, based on STAMP, result from **inadequate enforcement of control actions** triggered by the process model of the controllers, i.e. the feedback control process creates or does not handle properly dysfunctional interactions in the process, including interactions caused both by component failures and by system design flaws (Leveson et al., 2003).

Inadequate control actions can exist if all or some elements of the process control process have flaws. As stated in (Ishimatsu et al., 2010) *"because each component of the control loop may contribute to inadequate control, a classification of accident factors starts by examining each of the general control loop components and evaluating their potential contribution: (1) the controller may issue inadequate or inappropriate control actions, including inadequate handling of failures or disturbances in the physical process; (2) control actions may be inadequately executed, or (3) there may be missing or inadequate feedback. These same general factors apply at each level of the socio-technical safety control structure, but the interpretations (applications) of the factor at each level may differ."*

2.2 STPA

STPA is a hazard analysis based on the STAMP accident model. Its procedure consists of the following steps (Ishimatsu et al., 2010), (Leveson, 2004b):

- (1) Identify the hazards in the system that may allow accidents to occur and translate these into top-level safety constraints. In STAMP, the term **"hazard"** does not just imply the sudden release of energy or the presence of

a harmful agent, but a set of conditions, that together with other conditions in the environment, may lead to an accident.

- (2)
 - a) Create the **control structure**, that is, a functional diagram depicting the components of the socio-technical system together with their control and feedback paths. The control structure diagram may be revisited to depict more information as the analysis progresses.
 - b) Determine how hazards can occur. As mentioned above, hazardous states result from inadequate control actions in each of the system components that would violate the top level safety constraints. The inadequate control actions fall into the following four general categories:
 - i) A required control action to maintain safety is not provided.
 - ii) An incorrect or unsafe control action is provided that induces a loss.
 - iii) A potentially correct or adequate control action is provided too early, too late, or out of sequence.
 - iv) A correct control action is stopped too soon.
 - c) Restate the inadequate control actions as safety constraints.
- (3) Determine how the potentially hazardous control actions can occur and thus define how the safety constraints determined in Step (2) could be violated.
 - a) For each controller in the control structure, create a model of the process it controls. These models may contain information about the initial state of the controlled process, information on its current state or information about the state of the environment around the controlled process.
 - b) Examine the parts of the process control processes, within which each controller is embedded, to determine if they can contribute to or cause system level hazards. The task in this step is guided by a set of **generic control process flaws**, which are depicted in Figure 2.3. Each component of the control process may lead to an inadequate control action. Thus, each of the general control process components (i.e. the sensor, the controller, the actuator and the links between them) must be evaluated for their potential contribution to the hazardous control actions. For example, the controller may have incomplete

mental models about the controlled process; the feedback from the sensor to the controller may be missing or may be inadequate etc.

- (4) Restate any flaws identified in the previous step as safety constraints and repeat step (3). If necessary, revisit the control structure diagram in step (2) to depict new components or more detailed information on each identified component.

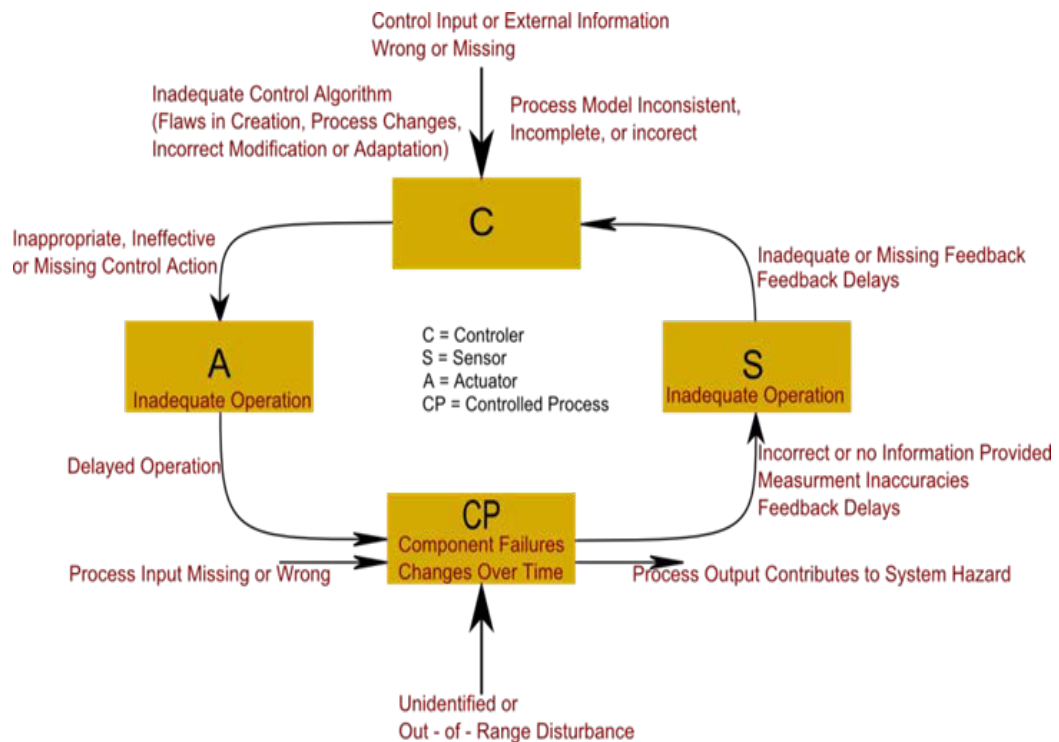


Figure 2.3: A feedback control process with generic control process flaws.

2.3 Justifying Perceivable Signs as Early Warnings

Any perceived data is classified as early warning sign by a controller if the controller possess models capable of relating it to a loss. Figure 2.4 depicts a model of the justification process of early warning signs where the concepts, shown with oval shapes, of early warning signs, causal factors and losses are connected via the relations "A" and "B". Relation "B" expresses that if an early warning sign is perceived by a sensor and comprehended by the controller, then a condition that is classified as a contributing factor to a loss is present in the system. The relation "A" expresses how this condition really contributes to a loss according to the process models of the controller.

When the controller, or in the case of a human controller, the human itself, believes, considers or assumes that the accident scenarios it possesses have clear and sound type "A" and "B" relations, then the justification of some observation data as early warning signs are effortless. When one, or both, of the "A" or "B" relations does not exist, then the observations, which are perceived by the sensor are not recognised as early warning signs by the controller. When the relations "A" and "B" exist but are not sound, then the perceived data may be classified as a weak sign.

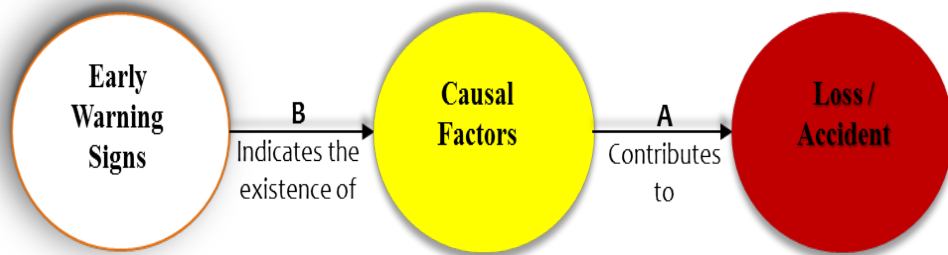


Figure 2.4: Early warning sign justification model.

Hazard analysis techniques produce models similar to that of Figure 2.4. Hazard analyses techniques, which are based on the sequential accident model, can detect only a subset of the type "A" relations. These hazard analyses can identify some detectable events, *which are warning signs and causal factors to a loss at the same time*. This subset is denoted with the link "A*" in Figure 2.5, which depicts a more detailed view of the early warning sign justification model shown in Figure 2.4.

However, when the issue being analysed is the safety of complex socio-technical systems, traditional hazard analyses are not able to provide a satisfactory description of accident scenarios due to the inherent limitations of sequential accident models. What is missing in this case is an additional set of "A" type relations as denoted by "A'" in Figure 2.5, which define those contributing factors to a loss that traditional hazard analyses techniques are unable to identify, such as structural and managerial deficiencies of the organisation and safety culture flaws, undesirable behaviours and interactions of system components, software flaws, system changes due to evolution and adaptation that affect safety.

To define the "A'" type of relations, it is necessary to utilise hazard analysis techniques, such as STPA, that are subject to systemic accident models. In essence, when the issue being analysed is the safety of complex socio-technical systems both "A'" and "A*" relations must be identified if it is to be

claimed that the accident models possessed by the controllers are as complete as possible. However, a limitation of STPA is that it lacks of steps which aim towards the identification of those perceivable signs which indicate:

- a) the presence of flaws in the feedback control processes of the system (i.e. the type B relation shown in Figure 2.5);
- b) the violation of its designing assumptions during the phase of operations.

One reason for this limitation could be that in some industries, system safety is viewed of, as having its primary role in development and most of the activities occur before operations begin (Leveson, 2012), paying thus less attention in identifying the early warning signs that may appear during the operation phase of the system.

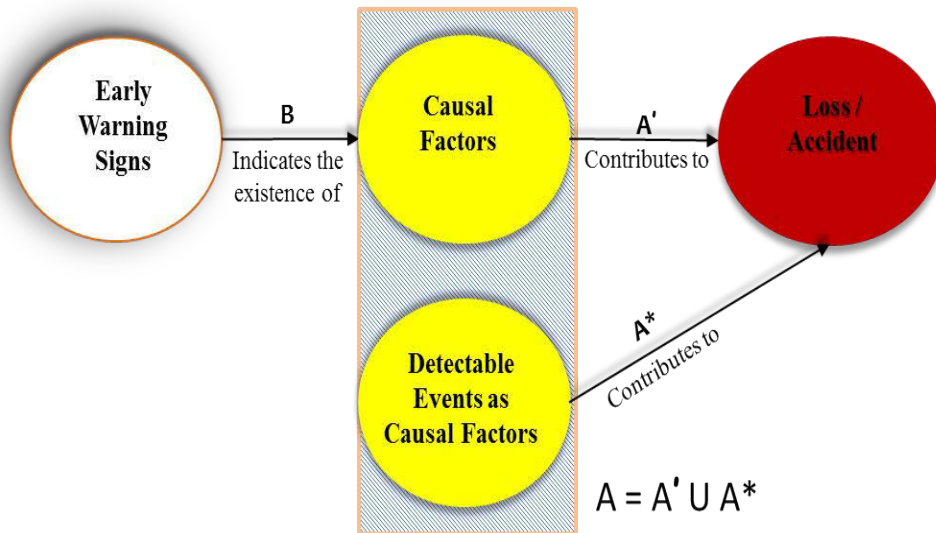


Figure 2.5: A more detailed view of the early warning signal justification model.

2.4 EWaSAP Steps

EWaSAP is an early warning sign analysis based on the STPA hazard analysis technique. The objective of early warning sign analysis is to help analysts to define models where a set of perceivable data (i.e. warning signs) indicating the presence of flaws in the control processes within the system, are associated with losses via accident scenarios. Referring to Figure 2.5, the aim of early warning sign analysis is to identify "as complete as possible", the "B" relations for a potential loss.

The EWaSAP steps are described below:

EW (1) For each flaw of the control process identified in Step 3(b) of STPA, identify the signs or signals that should or can be perceived by the sensor(s) indicating the presence of the flaw in the system.

EW (2) Describe the attributes of each warning sign. The rationale for the selection of these attributes is based on a set of interrogative questions such as:

- Source: *Where is the early warning sign arising from?*
- Receiver: *Which controller in the system should be informed about the presence of the flaw, which is comprehended via the perception of the sign?*
- Medium: *How was it is conveyed (means of data/information transfer) to a receiver?*
- Timestamp: *When was the information received?*

EW (3) Update the mental model of the controller(s) with reaction / adaptation rules so that to enforce the necessary changes to eliminate the detected flaw or to make the system resilient if the flaw cannot be corrected. (i.e. this step is an update to step 3(a) in the STPA process)

In Step EW(1), the analyst attempts to define the type B relations which are depicted in the justification model (see Figure 2.5). In Step EW(2) the attributes of the warning signs are defined. These attributes are characteristics or properties of the data, which when perceived by the sensor help the controller to justify that a contributing factor to a loss is present in the system. EW(3) the mental models of the controllers are updated with reaction or adaptation rules given the comprehension of the early warning signs.

2.5 Hazard and Early Warning Analysis

Aim of the hazard and early warning analysis is to identify "as complete as possible" sets of "A'", "A*" and "B" relations for a potential loss. This aim is achieved by extending STPA with EWaSAP to incorporate the type "B" relations shown in Figure 2.5 that connect early warning signs with causal factors to a loss. In particular, EWaSAP adds the early warning sign identification steps described in the previous section in between the 3rd and 4th steps of STPA. As result the steps of the hazard and early warning analysis derive by integrating STPA and EWaSAP steps in a single analysis. The logical sequence of the steps are shown in Table 2.1.

Table 2.1: The sequence of hazard and early warning analysis steps

Sequence of hazard and early warning analysis steps	Description	STPA and EWaSAP steps	
1	Identify the hazards	STPA Steps	1
2	Define the control structure diagram		2 (a)
3	Define for each feedback control process inadequate control actions		2 (b)
	Restate the inadequate control actions as safety constraints.		2 (c)
4	For each controller in the control structure, create a model of the process it controls		3 (a)
5	Define the flaws in each feedback control process using set of generic control process flaws,		3 (b)
6	For each flaw of the control process identified in Step 3(b) of STPA, identify the signs or signals that should or can be perceived by the sensor(s) indicating the presence of the flaw in the system.	EWaSAP steps	EW(1),
7	Specify the attributes of each warning sign		EW(2)
8	Update the mental model of the controller(s) created in step 3(a) of STPA with reaction / adaptation rules.		EW(3)
9	Restate any flaws identified in the previous step as safety constraints and repeat steps 3a 3b of STPA and steps EW1, EW2, EW3 of EWaSAP.	STPA step	4

2.6 Summary

This chapter described the STPA and EWaSAP and has explained their basic concepts. STPA is a systems theoretic hazard analysis approach whereas

EWaSAP is an early warning sign analysis approach. EWaSAP extends STPA in order to help the analysts identifying the early warning signs of each contributing factor to a loss that were identified with the use of STPA. When the STPA and EWaSAP are integrated they are forming a hazard and early warning analysis approach, which form the domain of the graphical DSMLs that are proposed as solution in this dissertation. As such, the key concepts, which were described in this chapter, will be represented by the constructs of the graphical DSMLs as it will be shown in a forthcoming chapter.

If everything seems under control, you're just not going fast enough.

Mario Andretti

3

Methods and Techniques for Domain-Specific Modeling

THE objective of this chapter is to describe important DSM concepts and techniques, such as Model Driven Development (MDD), alternative development strategies for graphical DSMLs and model transformation techniques. Emphasis is given on the distinction between DSML, where the main contribution of this research belongs to, and Domain Specific Programming Language (DSPL). It has been found that a DSMLs can be designed and developed with one of the following strategies: a) development from scratch; b) refining the concepts and relations of an existing DSML; and c) extending an existing DSML. Emphasis, thus, has been given on the description of the state of the art methods and tools for designing and developing a DSML from scratch and on the description of the Systems Modeling Language (SysML), which is an existing, UML based, DSML that provides models of systems engineering concepts and seemed of having the potential to be refined so that to develop the DSMLs for the problem domain of this thesis.

3.1 Model Driven Development

MDD (Selic, 2003), (Stahl et al., 2006) is an established and widely used approach to software development that aims at cost-effective automation and improved maintainability through abstract modeling (Pahl and Barrett, 2007). With this approach software development is treated as a set of transformations between successive models from requirements to analysis, to design, to implementation, to deployment (Thomas, 2004). MDD is about focusing on models rather than computer programs in software development (Sivonen, 2008). This makes the models easier to specify, understand, and maintain

(Selic, 2006). In MDD the models are the prime software artefacts and large parts of computer code may be generated from them. Custom, hand-written code is used to complete the functionality of the software, filling in those parts, which are not represented in the model (Kent, 2011).

In essence, the primary goal of MDD is to raise the level of abstraction of the problem domain, compared to programming languages such as JAVA, Python etc., by using concepts that are much less bound to the underlying implementation technology of a software but much closer to the problem domain. Thus any software developed using a MDD approach can be created, maintained and evolved more rapidly and efficiently compared to the typical software development process. In short, MDD aims to bridge the semantic gap between specification and implementation and as a result the various activities and tasks that comprise the software life cycle are simplified and formalised (Hailpern and Tarr, 2006).

The MDD framework consists of four layers as shown in Figure 3.1. The bottom level, M0, holds the user data, which are the actual data and objects that a user manipulates. The next level, M1, holds a model of the M0 user data. Level M2 holds a model of the information represented in M1. Because M2 is a model of a model, it's often referred to as meta-model. Finally, level M3 holds a model of the information represented in M2, and therefore it is often called the meta-meta-model. Each layer can be viewed independently of other layers and maintain its own design integrity (UML, 2003).

User defined graphical models of UML or DSML types are contained in layer M1 and are instances of a meta-modeling language contained into M2 Layer. For example, referring to UML models, the M2 level contains the UML meta-modeling constructs whereas the level M1 contains the user defined models that represent his problem domain. Referring to DSML, the M2 level contains the constructs of a meta-modeling language, such as SysML or Eclipse Modeling Framework (EMF) Ecore, whereas the level M1 contains the user defined domain specific models.

Several MDD approaches exist (Mattsson et al., 2009), such as Software Factories from Microsoft (MS) (Greenfield and Short, 2003), Object Management Group (OMG)'s Model Driven Architecture (MDA) (Miller et al., 2003), and DSM (Pitkänen and Mikkonen, 2006).

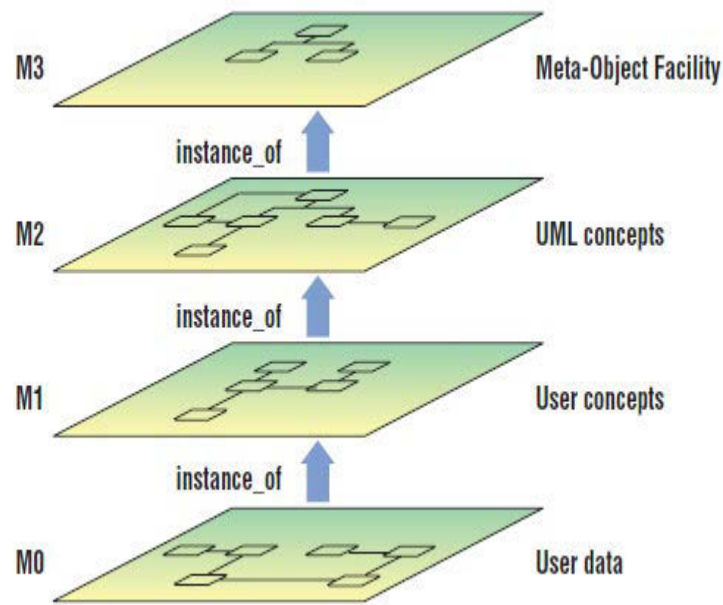


Figure 3.1: The four layer MDD architecture (Atkinson and Kuhne, 2003).

Software Factories focuses on product line development, that is similar but distinct software products (Demir, 2006). The mainstream MDD applications are tightly coupled with OMG's MDA, where the UML is typically utilised as a modeling language (Pitkänen and Mikkonen, 2006). The DSM MDD approach, on the other hand, focuses on providing specific solution of a problem domain by specifying a particular graphical DSML other than UML.

DSM is a MDD approach, which involves:

- (a) the systematic use of Domain Specific Language (DSL) to represent the various facets of a domain, in terms of domain models (Rivera et al., 2009), (Tranoris and Denazis, 2010);
- (b) the utilisation of a code generator **when this is necessary**.

One of the most frequently cited definitions of a DSL is that referred by (Van Deursen et al., 2000) which states that DSL is a "programming language or executable specification language that offers through appropriate notations and abstractions, expressive power focused on and usually restricted to, a particular problem domain".

There are two types of DSLs:

1. The Programming Languages (DSPL)
2. The Modeling languages (DSML)

The difference between DSPL and DSML is that the first is represented via a sequence of characters written and processed using an interpreter or compiler, while the second is represented via graphical models created usually in a canvas. For example HTML (webpage markup), SQL (database queries), Latex (text processing) or shell scripts of the UNIX systems are DSPLs, which allow experts with computer/programming skills to create web pages, query databases, write documents and scripts.

DSMLs, on the other hand, consists mostly of graphical models that represent domain concepts and relations and can be used by domain experts, which typically do not have programming skills. In practice, engineers express models using diagrams, structured text, and storyboards of one form or another (Thomas, 2004). Likewise, DSMLs are collections of shapes, such as boxes, circles, and lines. The domain concepts are represented by shapes. The relations are represented by linking lines. The shapes and the lines have a text associated to them that describe their roles and their editable properties.

When there is a need to transform user models to other models or languages, like, for example, when there is a need to generate code from a DSML, the framework shown in Figure 3.1 may be extended with one additional layer as Figure 3.2 shows. Specifically, in the case of code generation from a DSML the user defined graphical models created as result of the combined utilization of layers M0 to M3 of Figure 3.1 may generate code after the appropriate utilization of code generator facilities. In this case the generated code is hold in to the "Model transformation layer", which it is shown in the bottom of Figure 3.2 .

DSPLs and DSMLs are special-purpose languages (Spinellis, 2001) and not a general-purpose language (GPL) such as C or Java. Some of the main differences between DSMLs and GPLs are that a DSML focuses on a particular domain as stated previously, while GPL is a programming language designed to be used for writing software code in a wide variety of application domains. The use of DSMLs may or may not require programming skills, while the use of GPL requires programming skills.

This thesis emphasises on DSMLs and not on DSPLs and GPL. Therefore, the following sections will describe, the key tasks for designing and developing a DSML with code generation support.

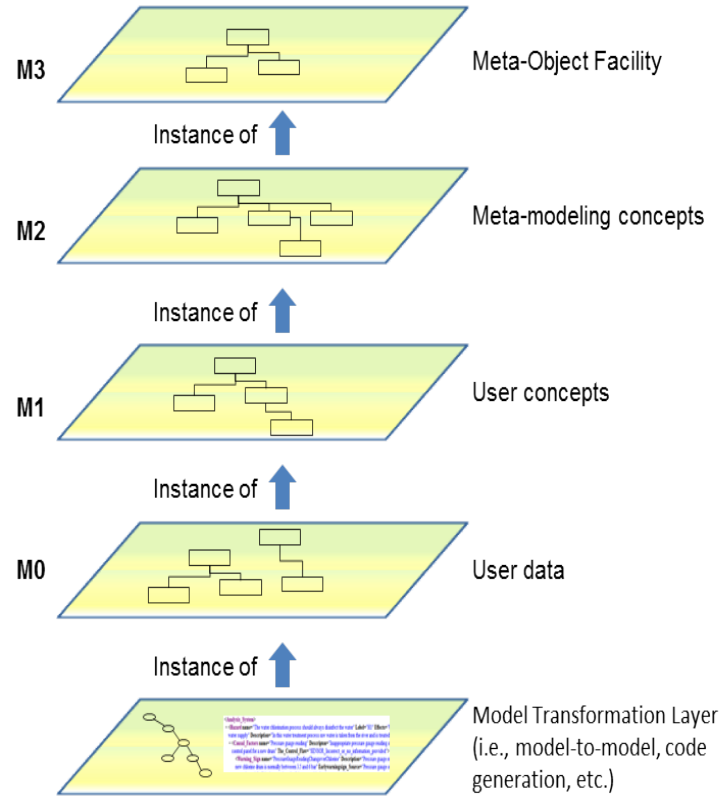


Figure 3.2: The model transformation layer as part of the generic MDD architecture.

3.2 Strategies for Defining DSMLs

There are three alternative strategies to define a DSML (Selic, 2007), (Karsai et al., 2009), (Selic, 2011):

1. Extend an existing modeling language, i.e. by supplementing new domain-specific concepts with new constructs in to the concepts of an existing language.
2. Refine an existing modeling language, by specialising some of the general concepts of an existing language.
3. Define a new modeling language from scratch (Kelly and Pohjonen, 2009).

When a DSML is created by extending an existing modeling language, the base concepts of the modeling language are extended further by adding additional modeling constructs for a specific domain problem. For example, in order to create a modeling language for a hazard and early warning analysis by extending an existing modeling language, a base modeling language

must be selected first. The base modeling language in this case should represent the appropriated STAMP and STPA safety related concepts in order to be considered as suitable. This strategy would be the ideal for the development of the hazard and early warning analysis DSML, if a DSML dedicated to STPA hazard analysis exists.

The refinement approach should be considered only if there is a significant semantic similarity and no semantic conflicts, such as contradictory constraints, between the concepts of the desired DSML and the concepts of the chosen base language (Selic, 2011). The UML is typically used as the base language in this development alternative because of the UML profile mechanisms, which allows the refinement of the UML constructs (Selic, 2007), (Silingas et al., 2009).

In essence, the difference between the refining and extending development strategies is that, in the later, the constructs of the final DSML are derived after extending the concepts of a base DSML with additional features and new concepts. Whereas, in the former development strategy a DSML do not need to be in existence but the constructs of a generic modeling language, such as UML, must be refined and/or extended using for example the UML profiling mechanism to construct the concepts of the final DSML.

A number of DSMLs have been developed based on the refinement strategy, such as the MARTE (MARTE, 2008) - a UML profile for real-time and embedded systems and SysML (team, 2008) - a UML profile designed for systems engineering.

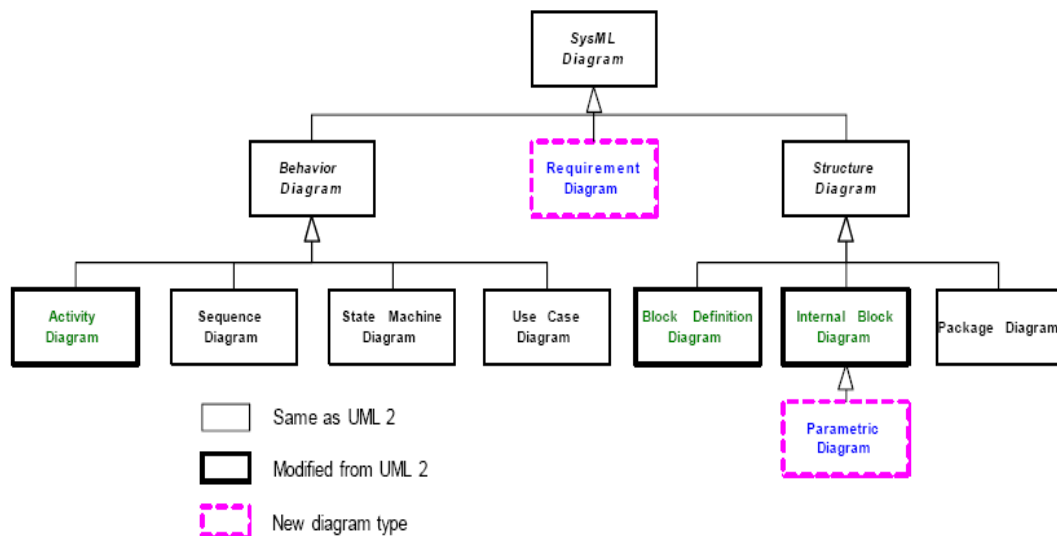


Figure 3.3: SysML taxonomy diagram (Boutekkouk et al., 2009).

For example, SysML uses seven of the UML 2.0's thirteen diagrams. Three UML2.0 diagrams, which are shown with the bold outlined diagrams in Figure 3.3, have been refined and two, that are shown in Figure 3.3 with the dotted outlines, have been added. Thus, there are nine diagrams in total into the SysML, which represent different concepts of the domain.

For instance, the "block definition" diagram represents the structural elements (i.e. physical or logical elements of the system, such as hardware, software, data, or persons) called blocks, and their composition and classification (Friedenthal et al., 2011).

The "Internal block" diagram represents the possible interconnections and interfaces between the parts of a block.

The "Activity" diagram represents the behaviour (i.e. represents the flow of inputs and outputs and the flow of control between activities) in terms of the order in which actions are executed based on the availability of their inputs, outputs, and control.

The "Requirement" diagram represents text-based requirements and their relationship with other requirements, design elements, and test cases for modeling the systems requirements, along with their traceability in relation to their architecture evolution (Espinoza et al., 2009).

Finally the "Parametric" diagram represents constraints on system parameter values , such as $F = m * a$, used to support engineering analysis (Friedenthal et al., 2011).

Furthermore, SysML inherits the stereotype construct by the UML , which is an extension mechanism. This allows to further extend and refine SysML concepts and create other DSMLs that need to represent concepts of the systems engineering domain.

3.3 Defining DSMLs From Scratch

To define a DSML from scratch, depending on its size and complexity, requires some consequent efforts and language design skills (Robert et al., 2009). However, it gives the developers the freedom to define those features in the DSML, which collectively may potentially provide maximum expressiveness to the end user.

When developing a DSML from scratch two main tasks should be accomplished. The first is to define its abstract syntax, and the second is to define its concrete syntax (Krahn et al., 2007) , (Oberortner et al., 2009) , (Ráth

et al., 2010). These steps are enhanced by special tools known as "meta-modeling tools". All these will be described in the following sections.

3.3.1 Abstract Syntax

The abstract syntax describes the vocabulary of the concepts which will be provided as constructs by the DSML and how they may be combined to create models. Referring to Figure 3.2, the abstract syntax define the constructs, that must be represented in to the M2 level of the generic MDD architecture. It consists of: a) the definitions of the concepts of the language, b) their relationships, c) a set of rules, which specify the semantics of the language.

In order to define the abstract syntax it is necessary to:

- i) Acquire and comprehend the domain knowledge;
- ii) Define the appropriate concepts, relations and rules of the language.

The first is achieved by an approach known as domain analysis and the second is achieved by defining the meta-model of the DSML.

3.3.1.1 Domain Analysis

Domain analysis seeks to identify and capture the commonalities and variabilities of the domain that is to be used to create the reusable language artefacts or domain models (Prieto-Díaz, 1990). The strategy that has been commonly used to identify the commonalities and the variabilities is to acquire first the knowledge of the domain and then to identify what is common and what is different in the concepts of the domain.

The knowledge on the domain is acquired from various sources, such as experts, technical documents, procedures, regulations and other materials (Imran, Foping, Feehan and Dokas, 2010a). The commonalities of the domain are captured by identifying the similar features, which are present into the instances of the concepts of the domain. The variabilities of the domain include those features that may differ among the domain concepts.

The output of a domain analysis can vary widely; however, it basically consists of domain specific terminology and semantics in more or less abstract form (Mernik et al., 2005). The approaches to conduct domain analysis can be mainly classified under the patterns of formal, informal and extraction from code (Mernik et al., 2005):

- Formal approach: The domain analysis approaches, which follow a methodology or a defined process, can be considered as formal approaches.

The use of formal patterns aims at ensuring that all of the relevant aspects of the domain are analysed and supporting the consistent analysis of the domain by following a specified procedure. Some formal domain analysis approaches include the: Feature-Oriented Domain Analysis (FODA) (Cohen et al., 1990), Sherlock (Valerio et al., 1997), Family-Oriented Abstraction, Specification and Translation (FAST) (Weiss and Lai, 1999), Domain Analysis and Reuse Environment (DARE) (Frakes et al., 1998).

- Informal approach: The domain analyses, which do not follow a methodology or a defined process are considered informal. Examples of the informal approaches includes the ontological approach (Tairas et al., 2009), the object oriented analysis approach and the use of Integration Definition for Function Modeling (IDEF0) (Imran, Foping, Feehan and Dokas, 2010b).
- Extraction from code: In this approach, the domain analysis is conducted by extracting the concepts from programming codes, that is, mining and extracting domain knowledge from existing computer programs written in general purpose programming languages, by means of manual or software supported methods.

3.3.1.2 Meta-Model

According to Kleppe (Kleppe, 2007), the meta-model represents the abstract syntax of the language. A meta-model is the embodiment of a modeling paradigm, that is, the set of axioms, notions, idioms, abstractions, and techniques that govern how systems within the domain are to be modeled (Ledeczi, Nordstrom, Karsai, Volgyesi and Maroti, 2001).

EMF Ecore is a meta-model specification language, which allows the development of meta-models. EMF Ecore is based on the implementation of the Essential Meta-object Facility (EMOF), which is a simplified version of the original OMG MetaObject Facility (MOF) standard. The main elements of EMF Ecore are depicted in Figure 3.4, which are the EClass, EAttribute, EReference and EDataType:

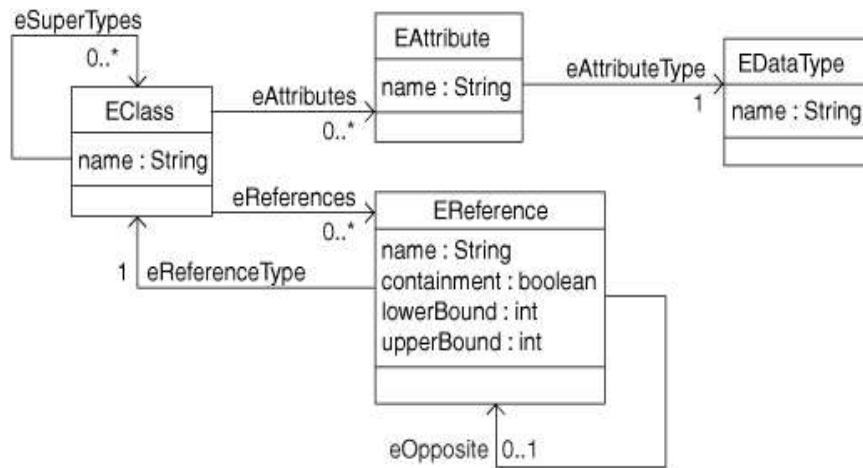


Figure 3.4: Ecore kernel (Budinsky and Brodsky, 2003).

EClass is used to represent a class of a model, it has an attribute to specify its name and can have zero or more attributes to express its properties and references;

EAttribute is used to represent the attributes of a model;

EReference represents an association between two classes (i.e. EClasses);

EDataType is used to provide the type of an attributes such as such as **EInt**, **EString** or **EDate** (Budinsky et al., 2003).

In addition to these, **EAnnotations** (Schauerhuber et al., 2006) can be used to provide additional information that cannot be represented directly via the Ecore-based meta-models into each element of the meta-model.

There are no particular terms on how the features of meta-models should be specified, as for a very similar modeling language a software developer may create different meta-models to represent the same domain concepts. Hence, the decision on which meta-model would be best for the development of a particular application relies on the expertise of the software developer (García-Magariño et al., 2009).

3.3.2 Concrete Syntax

Concrete syntax defines the notation that the end user will use to specify the models, which will conform with the abstract syntax. In essence, the concrete syntax provides the elements of the M1 level of the generic MDD architecture shown in Figure 3.2. Typical notation types are the textual and the

graphical. The advantage of a textual notation is that it is good at representing detail, while a visual notation is good at communicating structure (Clark et al., 2004).

In many problem domains, graphical notations are preferred by practitioners, as they often are the most intuitive representation of domain concepts (Esser and Janneck, 2001). Intuition, however, is a feature that is subjective. Each user may apprehend differently the true meaning of the concrete syntax of a DSML. As result, there is not any specific standard or criteria for ensuring that a DSML is intuitive¹. However, choosing notations that are standard or commonly used for each concept in the domain is suffice of addressing this problem, especially when the concrete syntax design takes place under the consultation of domain experts.

3.4 Model Transformation

Model transformation is an automated process that involves the models, which were created using a DSML, as inputs, and textual or other graphical models as outputs. The outputs of this process resides into the "Model transformation layer", which is shown in the generic MDD architecture of Figure 3.2.

There are three common model transformations approaches (Beydeda et al., 2005):

- 1) Refactoring transformations;
- 2) Model-to-model transformations;
- 3) Model-to-text transformations (i.e. code generation).

The refactoring transformation is performed when the information on a model is changed without altering its semantics. The output from this is a refactored model, which is a revision of the original model. A refactoring example is the renaming of all instances of a modeling construct where a particular entity name is used (Beydeda et al., 2005). This transformation is commonly used to improve the readability and extendibility of the models.

The model-to-model transformation (M2M) is performed when the user created models, (i.e. a UML model) must be converted in to another type of model (i.e. a Statechart model). Some of the common technologies used to perform model-to-model transformation are the: ATLAS Transformation Language (ATL) (Jouault and Kurtev, 2006), Epsilon Transformation Language

¹Eclipse Community Forums , a thread on "modeling constructs to be intuitive" at: <http://www.eclipse.org/forums/index.php/m/873160/>, last accessed (11th October, 2012).

(ETL) (Kolovos et al., 2008) and OMG's Query-View-Transformation (QVT) (OMG, 2005).

The model-to-text transformation (M2T), which is also known as model-to-code, is carried out when the information from user models converts to software code and other textual artefacts. The M2T transformation is commonly referred as code generation (Cuadrado and Molina, 2007). The transformation is not limited to programming languages such as Java and C, but also to the data definitions, deployment configuration, message schemas and other types of files.

3.5 Meta-modeling Tools that Support the Creation of DSMLs

Meta-modeling tools are special software programs, which enhance the development of DSML from scratch. DSML developers may define with these tools the constructs of the M0, M1, M2 levels, as well as the specifications of the model transformation layer that is shown into the generic MDD architecture of Figure 3.2.

The objective of the meta-modeling tools is to provide to developers the software libraries, which are necessary for defining the abstract and concrete syntax, as well as for the creation of the code generator facility. The meta-modeling tools speed up the development process of DSML from scratch. Without them the development process could take an enormous amount of time (Kelly, 2004), (Miotto and Vardanega, 2009).

The most commonly used meta-modeling tools are the Generic Modeling Environment (GME) (Ledeczi, Maroti, Bakay, Karsai, Garrett, Thomason, Nordstrom, Sprinkle and Volgyesi, 2001), MetaEdit+ (Kelly et al., 1996a), MS DSL tools (Cook et al., 2007), and Eclipse Graphical Modeling Framework (GMF)², which are introduced in the following sections.

3.5.1 Generic Modeling Environment

The GME was developed at the Institute for Software Integrated Systems at Vanderbilt University. It is a configurable toolkit for DSM and program synthesis environment, which has the ability to automatically transform requirements and design information into application software (Nordstrom et al.,

²Eclipse: Graphical Modeling Framework at: <http://www.eclipse.org/modeling/gmp/>, last accessed (11th October, 2012).

1999).

GME is used for creating and evolving DSMLs. It is configurable, enabling it to work with different domains. The configuration is accomplished through a UML based meta-modeling environment supported by a meta-modeling language called MetaGME (for Software Integrated Systems, 2005). The meta-model is used to automatically generate the target domain specific environment, which is then used to build domain models that are stored in a model database or in an XML format (for Software Integrated Systems, 2005), thereby specifying the domain specific modeling language for the application domain.

The two main components of GME are the GMeta and GModel. The GMeta is a graphical tool for constructing meta-models and the GModel implements the GME modeling concepts. Both GMeta and GModel demonstrate their services through graphical interface that is based on MS Component Object Model (COM)(Williams and Kindel, 1994) technology (Lédeczi, Bakay, Maroti, Volgyesi, Nordstrom, Sprinkle and Karsai, 2001).

3.5.2 MetaEdit+

MetaEdit+ is a commercial tool that provides an environment to support the design and development of domain specific modeling languages and has been applied successfully in various application domains. MetaEdit+ is configurable toolset that supports the development of different modeling languages by specifying their meta-model. MetaEdit+ employs the GOPPRR (Graph-Object-Property-Port-Role-Relationship)(Kelly et al., 1996b) as meta-modeling language based on which the meta-modes of the DSMLs, which each user of MetaEdit+ wants to create, are specified.

In MetaEdit+, the meta-model and its design model instances are stored into an object-oriented repository system, which supports complex references between design elements, e.g. inheritance and reuse by reference. The repository also enables multiple users to access and share the design data concurrently (Tolvanen et al., 2007). Furthermore, MetaEdit+ environment consists of two main components: MetaEdit+ workbench, for designing the modeling language by specifying its concepts, rules, notation and generator and MetaEdit+ Modeler, which provides the modeling tool functionality in creating and editing models such as programming editor, browser, generator³. The modeling language definition is stored in the MetaEdit+ repository,

³Available at <http://www.metacase.com/products.html>, last accessed (11th October, 2012).

from where it can be retrieved for later modification.

3.5.3 Microsoft DSL Tools

MS DSL is a suite of tools for creating, editing, visualising, and using domain-specific data for automating the enterprise software development process (Bézivin et al., 2005). MS DSL tools employ the Software Factories approach (Greenfield and Short, 2004) and its toolset relies on MS technologies. The DSL tool consists mainly of a project wizard on which a visual studio project template runs to collect the data that is needed to create a fully configured solution in which domain models can be defined. Furthermore, it consists of a graphical designer for defining and editing domain models specified using XML format. It also contains a set of code generators that takes as an input the source domain model definition and the designer definition and validates them.

3.5.4 Eclipse GMF

The GMF is the main initiative of the Eclipse community in its drive to provide support tools for DSM. The GMF project has been restructured into the GMF Tooling, GMF Runtime and GMF Notation projects under the Graphical Modeling Project (GMP)⁴. The GMF runtime provides a set of reusable components for graphical editors such as printing, image export, actions and toolbars. It aims to bridge the different command frameworks used by EMF (Steinberg et al., 2009) and Graphical Editing Framework (GEF) (Hudson, 2003), and allows graphical editors to be open and extendible.

The GMF tooling offers a model-driven approach to generating graphical editors in Eclipse, based on the GMF runtime. Both GMF runtime (i.e. runtime infrastructure) and GMF tooling (i.e. generative component) are used for developing graphical editors based on the EMF with the use of Ecore meta-modeling language and the GEF (Di Ruscio et al., 2011). The GMF notation provides a standard EMF notational meta-model, which is a standard means for persisting diagrams information separately from the domain model.

In essence, GMF forms a generative bridge between EMF and GEF, whereby a diagram definition will be linked to a domain visual language model, which

⁴Available at <http://www.eclipse.org/modeling/gmp/?project=gmf-tooling#gmf-tooling>, last accessed (11th October, 2012).

serves as an input to the generation of a visual editor (Taentzer, 2006). Arguably, GMF can be considered as the mainstream approach to facilitate the tooling environment for domain specific graphical editor development within the Eclipse platform (Di Ruscio et al., 2011).

3.6 Summary

This chapter has introduced the key concepts of DSM. A DSM development approach involves the creation of some type of DSL, such as DSPL or DSML, and in some cases the inclusion of model transformation capabilities, such as code generation. The differences between DSPL and DSML were presented. These differences are: a) DSPLs have a textual form whereas DSMLs have a graphical form, b) DSPLs are intended to be used by users with programming skills, whereas DSMLs are intended to be used by users without programming skills.

Three common types of model transformation were identified in the literature and introduced herein. These are: a) Refactoring transformations; b) Model-to- model transformations; and c) Model to text or code generation.

Three DSML development approaches were identified. The goal in one of them is to extent an existing DSML. The goal of the second is to refine an existing modeling language (i.e. usually the UML). The aim of the last approach is to create a DSML from scratch. **Thus, possible alternatives in addressing the research problem in this thesis are the following:**

- a) To extent an existing STPA based DSML;
- b) To refine the SysML meta-model so that to include the concepts of the hazard and early warning analysis. The rational for assessing this as an alternative solution to the problem is based on the fact that STPA and EWaSAP are based on a systems theoretic accident model;
- c) To define a DSML for hazard and early warning analysis from scratch.

Two are the main tasks that should be accomplished when developing a DSML from scratch. The first is to define the specifications of DSMLs abstract syntax and the second is to define its concrete syntax. The abstract syntax specifies the vocabulary of concepts provided by the DSML. The concrete syntax defines the notations that the end user will use to specify his programs with the DSML. Finally, the chapter has presented the state of art tools for designing DSML from scratch. These tools are known as "meta-modeling tools" and include the GME, MetaEdit+ , MS DSL tools, and Eclipse GMF.

Safety is defined absolutely as a quality that may not be entirely achievable, but that can still be defined in absolute terms as a desirable quality that can be improved.

Nicolas Dulac

4

A Review of DSMLs for Early Warning Systems

THIS chapter begins by listing a number of DSMLs, which have been used in a variety of domains, such as telephone and radio services and computer games. Emphasis then is given to a number of DSMLs, which were used for the creation of domain specific hazard analysis models and for the development of early warning systems. That resulted in studying four DSMLs, namely the SOPHIA, EAST-ADL, Stream-Oriented DSML, Open-COM and Transition Diagrams DSML, in more detail. These DSMLs are then assessed for their potential use in creating the risk knowledge elements of early warning systems. The criteria for the assessment were: 1) the utilisation of domain concepts that conforms to accidents and hazard analysis such as, "hazard", "causal factor", "warning"; and 2) their capability of generating code. Two out of the four DSMLs were found of being capable of generating code. Both of them were created from scratch. The result of the literature review has been the confirmation of the nonexistence of a DSML dedicated to STPA hazard analysis. Thus, one alternative strategy for addressing the research problem, that of developing the DSML for the hazard and early warning analysis by extending an existing DSML, was eliminated.

4.1 Examples of DSMLs Application Domains

DSMLs are used in a vast array of domains to create applications for a broad collection of languages and platforms, including many industrial applications (Imran, Dokas, Feehan and Foping, 2010). Examples of domains in which DSMLs have been developed and used include communications (Claypool et al., 2009), mobile health monitoring (Balagtas-Fernandez and Hussmann,

2009), architectural knowledge capture (Olumofin and Mišić, 2006), and computer games (Furtado and Santos, 2006).

Examples of applying DSML are also given in (Kelly and Tolvanen, 2008). Each example examines the use of DSMLs for different problem domains, ranging from insurance products, home automation, IP telephony and call processing, mobile phone applications and digital wristwatch to microcontroller applications. The above examples indicate that DSMLs have reached a sufficient level of maturity that contributed positively in their adoption for the creation of a significant number of real word applications.

4.2 DSMLs With Hazard Analysis Concepts

The domain of hazard analyses use concepts such as "Hazard", "Cause", "Effect", "Consequence". General purpose graphical modeling languages, such as UML, can be used to model these concepts. For example, the UML profile mechanism can be used by the developers to "define" these concepts in order to design and develop a DSML. Two DSMLs have been found in the literature that formally express hazard analysis concepts in this way. These are the SOPHIA and the EAST-ADL.

4.2.1 SOPHIA

SOPHIA (Cancila et al., 2009a) is a DSML that formalises safety related concepts and their relationships with system modeling constructs. SOPHIA's language specification derives from UML via a refinement approach.

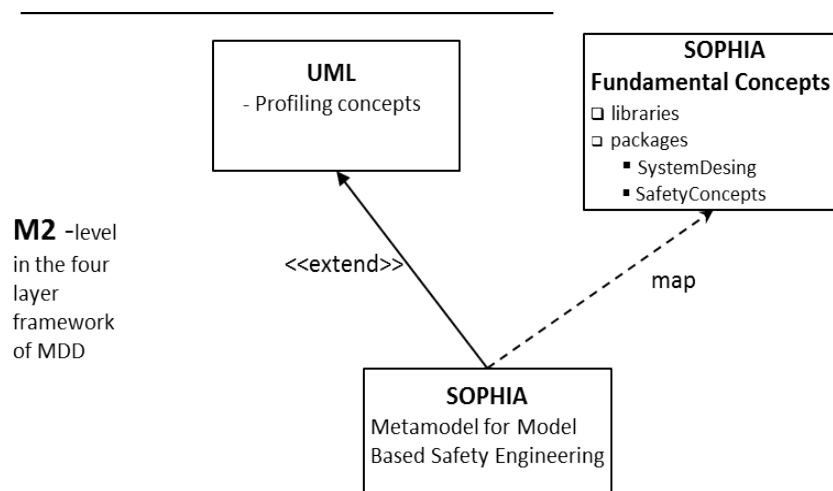


Figure 4.1: The refinement of the UML meta-model with the fundamental concepts of SOPHIA.

Figure 4.1 shows the "infiltration" of SOPHIA fundamental concepts in to the UML meta-model, with the use of the UML profiling mechanism. In essence, the fundamental concepts of SOPHIA have refined the UML concepts via the use of the UML profile mechanism.

The fundamental concepts of SOPHIA are the "packages" and the "libraries". The "packages" define the notions of the language and their relationships, whereas the "libraries" specify the data types of the elements contained in the "packages". There are two subpackages in the "packages" of SOPHIA. The first is called "SystemDesign" and specifies the relationship between the system concepts and the model element of the system. The second is the "SafetyConcepts", which contains the packages "Accident", "Mitigations" and "FaultContainmentRegion". The "Accident" package defines the notions and relationships involved in an accident, such as "Hazard", "AccidentCase" and "AccidenConsequences". The "Mitigations" package defines the notions and relationships about the mechanism, which is responsible to mitigate an accident. The "FaultContainmentRegion" package defines the concepts and relationships, which are involved during any error propagations efforts.

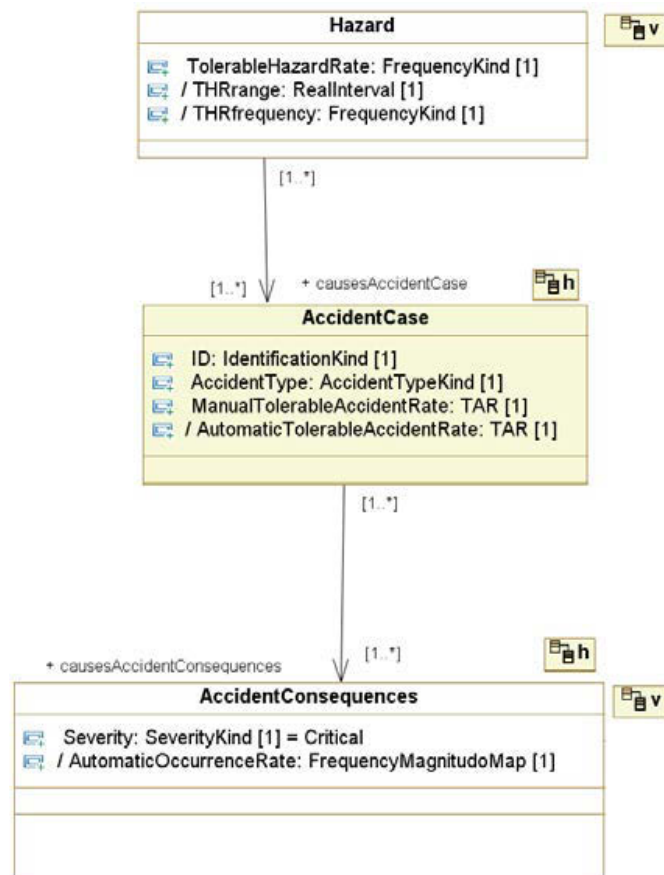


Figure 4.2: A SOPHIA model with elements from the "Accident" package (Cancila et al., 2009a).

Figure 4.2 shows some elements of the "Accident" package, that were used to define the concept of "TolerableAccidentRate", whose definition in the problem domain, where SOPHIA was used (i.e. the railway safety engineering), is "a threshold between what is tolerable and what is undesirable with respect to the consequence of an accident". The tolerable accident rate in this example is an attribute of the concept "Hazard". The concept "Hazard" represents "an event observable at the system boundary, which has potential either directly or in combination with other factors (external to the system), for giving rise to an accident at railway system level" (Cancila et al., 2009a). Each accident leads to one to many unintended events with undesirable outcomes represented with the concept of "AccidentCase". An "AccidentCase" can have the following properties: a unique "ID"; an "Accident-Type" chosen from a statically pre-defined list; "ManualTolerableAccidentRate" and "AutomaticTolerableAccidentRate". Each "AccidentCase" leads to one to many "AccidentConsequences". The "AccidentConsequences" represent the adverse result of a given hazard. An attribute of the "AccidentCase" is the "Severity", which may take only one of the following four predefined values: "Catastrophic", "Critical", "Marginal", or "Insignificant".

The M1 level of SOPHIA, that is the elements of its concrete syntax that a user can use to create his models, is composed of the graphical representations of SOPHIA's concepts and of some SysML concepts, such as the "requirement diagram" and the "block diagram". The SysML concepts act as complementary to those of SOPHIA. Thereby, SOPHIA and SysML, which are both designed by refining the UML concepts using the UML profile mechanism, can be used jointly by the analysts via the same UML editor (Cancila et al., 2009a), as Figure 4.3 shows (Cancila et al., 2009b).

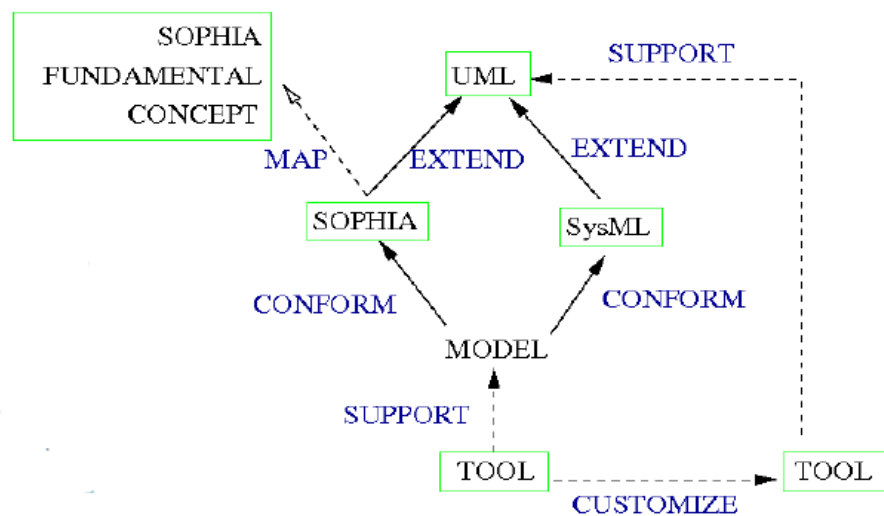


Figure 4.3: Sophia and SysML integration model.

In essence, a number of basic concepts coming from the domain of hazard analyses are embedded into SOPHIA's meta-model definition. However, there is no indication of the existence of the early warning sign concept. Furthermore, the publications describing SOPHIA lack of discussions on the accident model, which was adopted in order to formalise the safety related concepts. It seems, however, that the developers of SOPHIA have adopted a traditional and not a contemporary accident model.

4.2.2 EAST-ADL

EAST-ADL (Electronics Architecture and Software Technology-Architecture Description Language) is an Architectural Description Language that provides a means to model and analyse software architectures, which are required to sustain the evolution of vehicle electronics. Its elements can be used to enhance the hazard analysis process carried upon the features of the electronics and software in the vehicles. Figure 4.4 depicts how a number of hazard analysis concepts, such as "Hazard" and "HazardousEvent" were defined in the meta-model level (i.e. Level M2 of the architecture) of EAST-ADL architecture.

In general, the "Hazards" are identified by negating the functionality of an "Item". Risk assessment is done on a set of "Hazardous Events". The "Hazardous Event" is modelled as a "Hazard" in combination with a given "operationalSituation". The "operationalSituation" in EAST-ADL is modelled as a combination of "traffic" and "environmentSituations" and a "UseCase" of the "Feature" corresponding to the "Item". A "SafetyGoal" could be seen as a "SafetyRequirement" on the vehicle level.

Looking at the elements of EAST-ADL meta-model it can be concluded that this DSML is heavily focused on the creation of accident scenarios that may arise only by component failures, which according to the systemic accident models such as STAMP, are only one contributing factor to accidents (Leveson, 2012). Other, contributing factors to accidents in complex socio-technical systems according to STAMP that need to be taken into account are organisational and social factors, human performance and software design flaws (Leveson, 2004b).

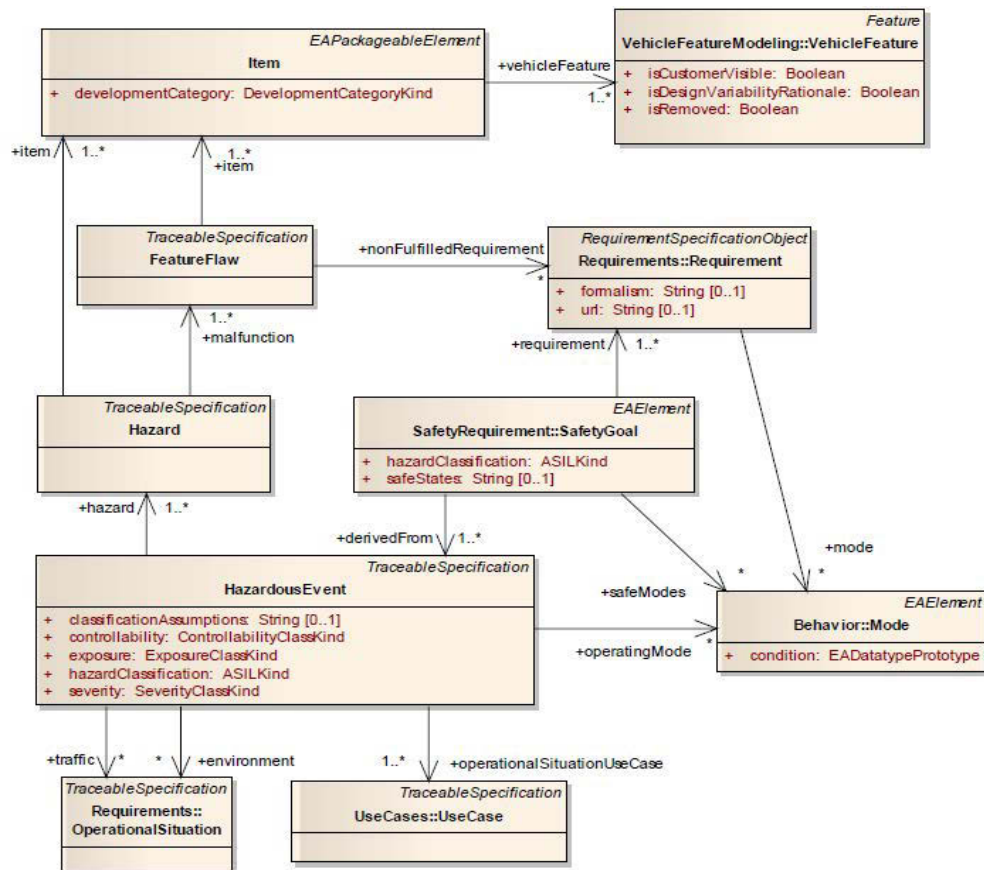


Figure 4.4: Snippet of meta-model specifications for EAST-ADL.

4.3 DSML for Risk Knowledge Modeling of Early Warning Systems

Two DSMLs capable of generating code so that they can be used in early warning systems where identified. The Stream-Oriented DSML for earthquake detection algorithms and the OpenCOM and Transition Diagrams DSML for the development of flood warning early warning systems. Both were developed from scratch.

4.3.1 The Stream-Oriented DSML

The purpose of the Stream-Oriented DSML is to generate earthquake detection algorithms from models that can run on the nodes of wireless sensor networks (WSN) in order to process the data received by the sensors. The generated algorithms are set up to work so that, if an earthquake is detected by one node, this information can then be passed to the surrounding nodes, and once there is consensus that an earthquake event is occurring, a warning signal will be disseminated throughout the whole network.

The main concepts used in a Stream-Oriented DSML are the: "Source", "Filter" and "Sink". The "Source" represents the sensor reading streams. The "Filter" is responsible for passing only those sensor readings that are considered to be the beginning of an earthquake and blocks all others. To manage the time interval of an earthquake detection warning at a reasonable level the idea of "time filter" has been employed. The "Sink" is responsible for generating an earthquake detection warning, for example, by activating a warning alarm. To control the sound level of the warning, a corresponding parameter named "SoundLevel" was included. The meta-model specification of the Stream-Oriented DSML (i.e. the Level M2 of its architecture) is shown in Figure 4.5.

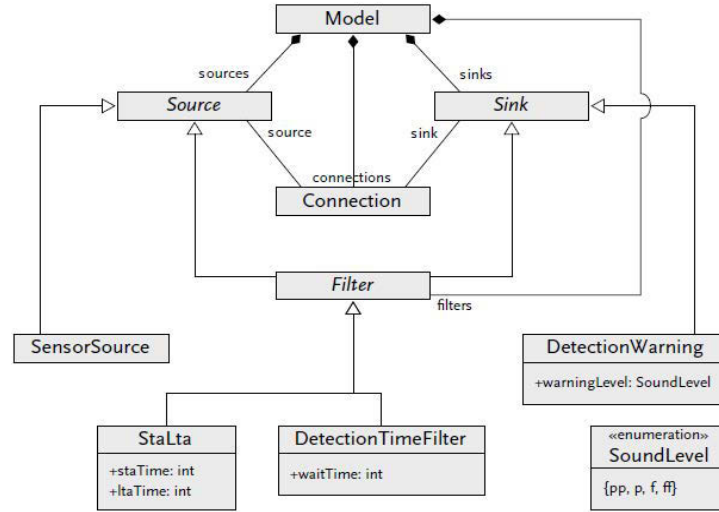


Figure 4.5: Meta-model specification of Stream-Oriented DSML (Sadilek, 2007b).

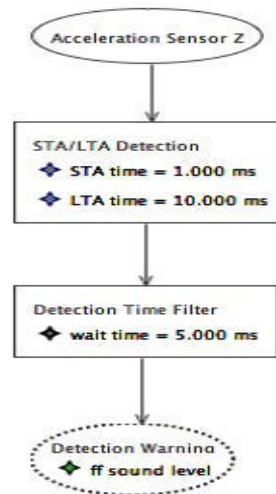


Figure 4.6: A user defined earthquake detection algorithm model (Sadilek, 2007a).

A user generated model using the concrete syntax of the Stream-Oriented DSML (i.e. the elements of the Level M1 of its architecture) is shown in Figure 4.6.

The Eclipse Integrated Development Environment (IDE) (Holzner, 2004) was utilised for the development of the Stream-Oriented DSML. The meta-modeling tool, which was used to create this DSML is the GMF. The approach for the M2T transformation in this DSML consisted of two sub-tasks: a) a syntactic translation of a program from its meta-model-based representation to its targeted code representation, the Scheme (Dybvig, 2003) in this case, which is a functional programming language and one of the two main dialects of the programming language Lisp (Van Rossum and Drake, 2003), and b) a semantic translation of the domain-specific concepts to base concepts of Scheme and the target platform by means of Scheme's abstraction facilities. The approach, which was used in this DSML achieves the same results compared to MDA but in a different order of task execution (Sadilek, 2007a).

4.3.2 The OpenCOM and Transition Diagrams DSML

An adaptive system is capable of changing behaviour at runtime and exhibit dynamic variability or runtime variability. In this context, there are two types of dynamic variability namely the Environment or Context variability and the Structural variability. The OpenCom DSML and the Transition Diagrams DSML are two DSMLs that form a set based on which the dynamic reconfiguration of adaptive systems may be achieved. These DSMLs are brought together via a dedicated editor called Genie (Bencomo, Grace, Flores, Hughes and Blair, 2008b). Genie has been implemented using the MetaEdit+ meta-modeling tool.

The meta-model of the OpenCom DSML contains definitions of the concepts, which are necessary for the modeling of the structural variability of adaptive systems. The models of the OpenCom DSML define the resulting architectural configurations. The meta-model of the Transition Diagrams DSML defines the necessary concepts to model the environment variability or context variability of adaptive systems. The models created by the Transition Diagrams DSML define the conditions under which a system must adapt.

The meta-model of OpenCOM is shown in Figure 4.7. Its main concepts are: "OpenCom Component", "Capsule", "Interface", "Binding Component" and "Component Framework". The "OpenCOM Component" represents a reusable software component. Each "OpenCOM Component" component is

related to an "Interface". The "Interface" allows an "OpenCOM Component" to provide and receive some (necessary) services to and from other "OpenCOM Components". The "Binding Component" defines the possible associations of the "Interface" of a component. The "Capsules" are the elements responsible for loading the "Binding Component" and the "Interface" of an "OpenCom Component" in runtime. The "Component Framework" is a set of components that cooperate to address a required functionality or structure (e.g. service discovery and advertising, security etc).

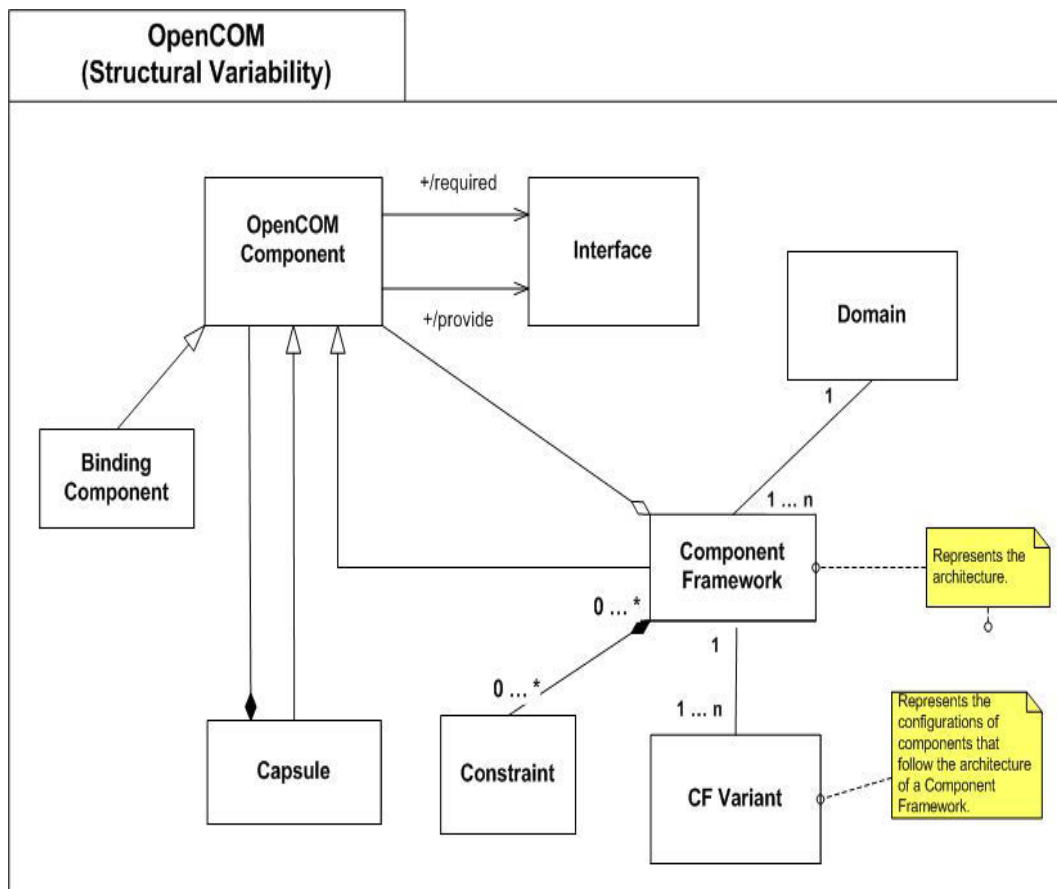


Figure 4.7: OpenCom meta-model (Bencomo, 2008).

The meta-model of the Transition Diagrams DSML is shown in Figure 4.8. The basic concepts defined in this meta-model are the "Structural Variant", "Transition" and "Trigger". The "Structural Variant" represents a possible configuration of the "Component Framework". A "Transition" expresses the change of the "Structural Variant" from the current state to a new state each time a "Trigger" is activated. "Triggers" define the existing conditions in the environment that may force a change in the system.

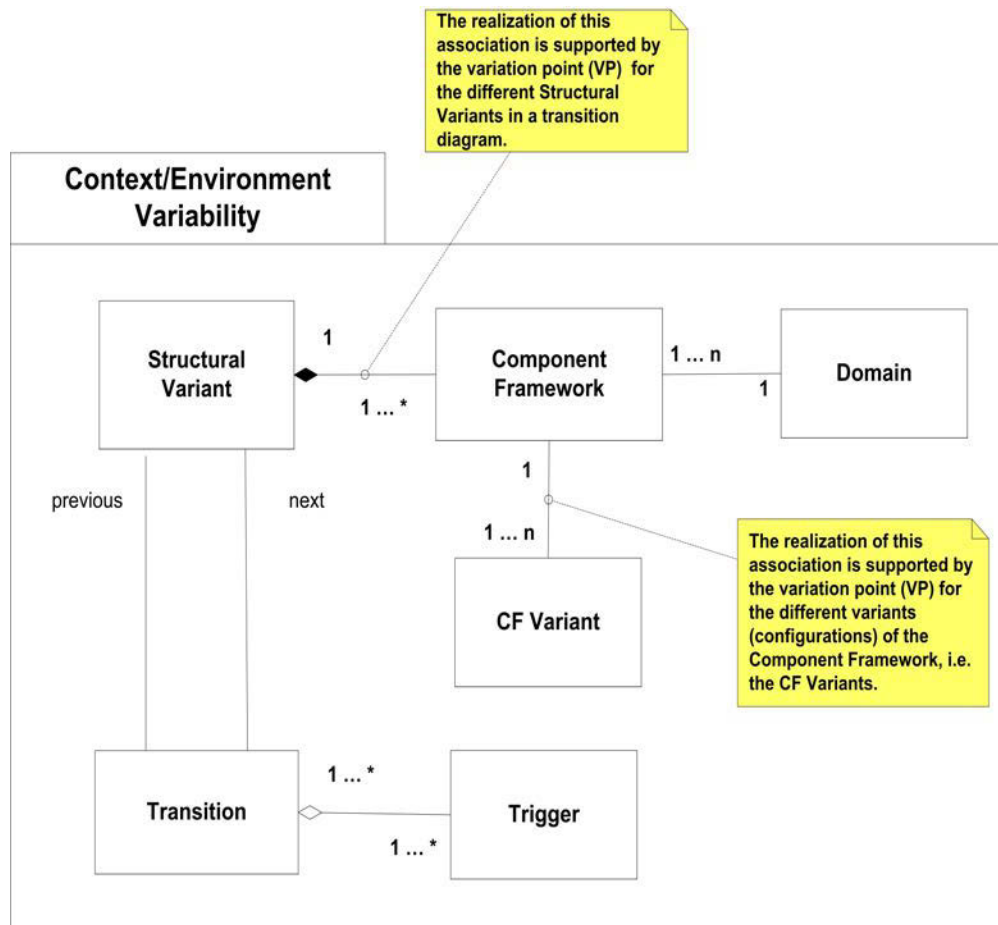


Figure 4.8: The meta-model to model the context and environment variability (Bencomo, 2008).

Using the modeling constructs of these DSMLs the developer can specify the configurations of the components and their transition diagrams. The model transformation capability of these modeling languages allows different software artefacts to be generated via model transformations. With the OpenCom DSML, the developer can define the components and their configurations, which all together define the component frameworks. Whereas, the reconfiguration policies of the adaptive system can be generated from the Transition Diagrams DSML.

The use of Genie is demonstrated by a case study on flood warning system (Bencomo, Sawyer, Blair and Grace, 2008). GridStix is a grid-enabled wireless sensor network for flood management. The model based approach supported by Genie consists of three different levels of abstractions (abstraction levels are raised from bottom to top) as shown in Figure 4.9 (Bencomo, Grace, Flores, Hughes and Blair, 2008b). The Level 1, is populated by different software artefacts like component source code, and files of configurations of

component frameworks and reconfiguration policies. The Level 2 in the middle is populated models associated with components and component frameworks (configurations). These models provide visual representations of the component configurations and are constructed using OpenCOM DSML. The Level 3 at the top is populated with the specification (models) of adaptations, where these models are in essence transition diagrams that guides the reconfiguration and adaptation process of GridStix. These models are specified using Transition Diagrams DSML.

In the case study the purpose of the two DSMLs was to generate an adaptive, to environmental conditions, dissemination and communication mechanism for the early warning system. The communication mechanism should be able to activate different channels, such as Wifi, Bluetooth for communicating the warnings and the alerts to the appropriate stakeholder, as Figure 4.9 shows, depending on the water fluctuation and/or flood level.

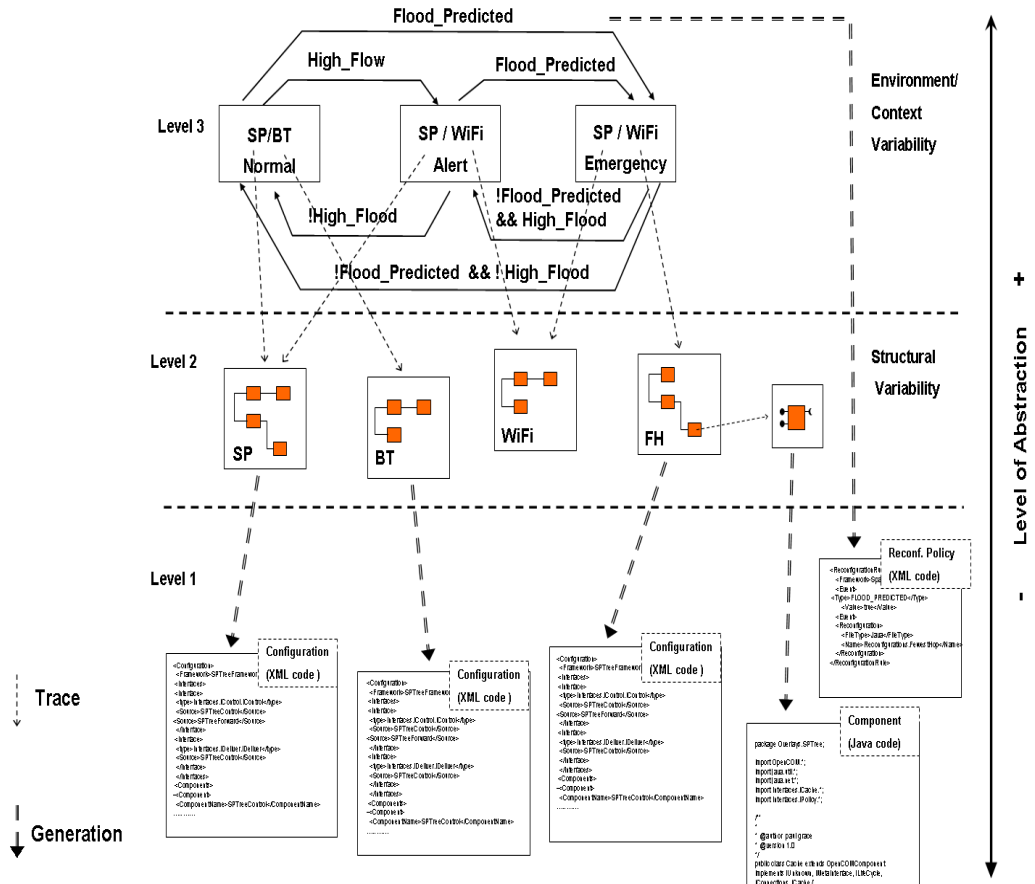


Figure 4.9: Overview of the approach implemented by Genie (Bencomo, Grace, Flores, Hughes and Blair, 2008a).

4.4 Discussion and Conclusions

The search for DSMLs, which are capable of modeling safety related concepts, resulted in the identification of four cases. These are: 1) SOPHIA, 2) EAST-ADL, 3) Stream-Oriented DSML, 4) OpenCOM and Transition Diagrams DSML.

SOPHIA and EAST-ADL can model the cause and effect relations that may lead to component failures (i.e. in the case of EAST-ADL) and accidents (i.e. in the case of SOPHIA). Both were developed through the refinement approach and used the UML profiling mechanism to define their concepts and relations into their meta-model level. However, they are not providing code generation facilities and, thus, they cannot produce executable code to be used into the risk knowledge element of early warning systems.

The users of Stream-Oriented DSML can model earthquake detection algorithms. They can, in a sense, model the causes, which may trigger earthquakes. Stream-Oriented DSML was developed from scratch using the open-source eclipse based GMF meta-modeling tool. It has a code generation facility and it can generate executable code that can be used to form the risk knowledge element of earthquake early warning systems. However, the constructs of this language represent concepts that belong into the natural hazards domain and are not the same as the concepts of the accident models that explain the phenomenon of accidents in manmade systems. Thus, the concepts and relations of this DSML are not capable of representing the concepts of the contemporary hazard analysis techniques that are used to create accident scenarios of manmade systems.

The OpenCOM and Transition Diagrams DSMLs are two DSMLs that have been integrated into a dedicated DSML editor called Genie. Both DSMLs were developed from scratch using the MetaEdit+, which is a commercial meta-modeling tool. Genie provides code generation mechanism, thus, these DSML are capable of creating executable code. In fact, they were used to model the necessary adaptations of a flood early warning system given as inputs the dynamic changes of water fluctuations. However, the purpose of these DSMLs is to provide an adaptive dissemination and communication mechanism for the flood early warning system and not the models for its risk knowledge element.

SOPHIA is a DSML, which refines UML concepts and makes use of some SysML concepts so that to generate safety related concepts. As result, the users of SOPHIA can create models of accident scenarios. Thus, the strategy of extending SOPHIA for developing the early warning analysis DSML

seemed feasible. However, this alternative falls short according to the criteria mentioned previously in this chapter about the need of the hazard and early warning analysis DSML to:

- 1) Support the creation of models based on contemporary accident models;
- 2) Provide code generation support.

Indeed, the accident model, which has been adopted for the creation of SOPHIA's meta-models, accord with traditional and not with any contemporary accident model. Furthermore, SOPHIA is not equipped with code generator mechanisms. From the literature review, it became apparent that another feasible way of developing the hazard and early warning analysis DSML is by refining the UML by using the UML profile mechanism in a manner similar to SOPHIA and SysML, but with the integration of code generation. There were two challenges in this alternative. The first was that UML profiles were penalised by lacks of methodological guidelines and tool support (Robert et al., 2009) for designing DSMLs and standardised UML profile generation process (Giachetti et al., 2009). The second challenge was to find alternative ways of generating code from the modeling constructs of the hazard and early warning analysis DSML, which will be specified by the UML profile mechanism. After conducting a literature search about this matter it was made clear that the only alternative was to develop a dedicated code generator, similarly to the work of (Silingas et al., 2009) (Vidal et al., 2009).

Thus, the alternative ways of achieving the research objective have been limited to the following two:

- 1) Create a DSML UML profile generation process together with a dedicated code generator capable of supporting the UML profile mechanism and based on these define the proper meta-modeling concepts of the hazard and early warning analysis in UML.
- 2) Create a hazard and early warning analysis DSML from scratch.

It was assessed that by selecting the first alternative, the entire research effort would have been exhausted in creating a code generator for UML profiles and that is actually something, which deviates completely from the initial research objectives. As a result, a decision was made to develop the early warning analysis DSML from scratch. This decision enforced the identification of the appropriate open source tools and frameworks which will be presented in the following chapter.

The first rule of any technology used in a business is that automation applied to an efficient operation will magnify the efficiency. The second is that automation applied to an inefficient operation will magnify the inefficiency.

Bill Gates

5

Software Technologies Used for the Development of the Hazard and Early Warning Analysis DSML

THE previous chapter described existing DSMLs that can model a number of hazard analysis concepts. It described also the rational based on which it was decided to create the hazard and early warning analysis DSML from scratch. This chapter describes the open source software technologies, which were selected to design and develop the early warning sign analysis DSML from scratch. Specifically, this chapter describes software technologies such as Java, Epsilon Generation Language (EGL), Epsilon Validation Language (EVL), XML and the modeling frameworks EMF and GEF, which were "plugged in" to the GMF meta-modeling tool for the development the hazard and early warning analysis DSML.

5.1 The Development Platform

Eclipse is an extensible platform structured around the concept of extension points, which are software programs for extending the platform by contributing additional functionalities that are commonly referred as plug-ins (Sivonen, 2008). For example, the Eclipse plug-in that was used for the creation of the hazard and early warning analysis DSML is the GMF, a detailed description of which was provided in section 3.5.4.

Eclipse IDE offers advanced features to the developers, such as auto code completion for long variables or method names, helpful keyboard short cuts,

real time compilation for finding errors like wrong input parameters and undeclared variables, real time compilation that provides indications for finding errors or potential errors such as undeclared variables, unmatched brackets, and wrong input parameters or return values, and code-refactoring to increase code clarity and maintainability by restructuring (Chen and Marx, 2005).

The architecture of Eclipse is presented in Figure 5.1. It is composed of three layers (Gamma and Beck, 2004): 1) Platform; 2) Java development tools (JDT); and 3) Plug-in development environment (PDE). The platform is the bottom layer, which its purpose is to define the common infrastructure of Eclipse. The Java development tools represent the middle layer. This layer adds a full feature Java IDE to Eclipse. Finally, the plug-in development environment is the top layer, which extends JDT in order to support the development of plug-ins into Eclipse.

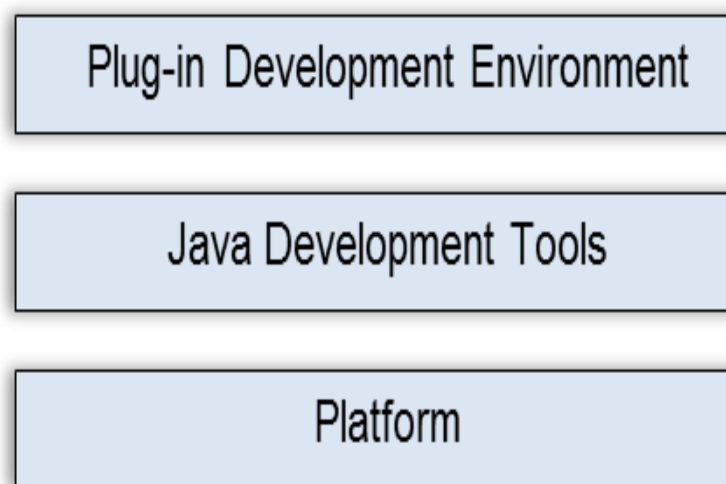


Figure 5.1: The three layers of eclipse architecture (Gamma and Beck, 2004).

5.2 The Main Components of the GMF Framework

GMF contains two main modeling frameworks. These are the EMF and the GEF. The EMF is used to manage the meta-model (i.e. the Level M2) of DSMLs and the GEF is used to create the graphical representation (i.e. the Level M1) of the DSMLs. Both frameworks are described in the following sections.

5.2.1 Eclipse Modeling Framework

EMF (Budinsky, 2004) (Moore et al., 2004) is a modeling framework¹ for MDD. It has evolved as one of the standard technologies which are needed to define a DSML and to create Eclipse based applications with model transformation capabilities. The EMF consists of three fundamental parts (Colombo et al., 2009):

- 1) The Core framework that includes a meta-model for describing models and runtime support for change notification and XMI serialisation;
- 2) The Edit framework that includes generic reusable classes for building editors for EMF models;
- 3) The Codegen framework that provides code generation facilities to build a complete editor for an EMF model.

The meta-model contained in the EMF is called EMF Ecore or simply Ecore. Ecore models can be created by different means such as, by transforming Java notations, UML models and XML schema. It also provides tools and runtime support that produce a set of Java implementation classes from the specified meta-model. These classes are extensible, regenerable and can be modified by adding user defined methods and instance variables. The user defined modifications to the implementation classes can be retained even when the model changes and implementation classes are regenerated².

5.2.2 Graphical Editing Framework

GEF (Moore et al., 2004) is an open source framework³. Its purpose is to provide all necessary libraries for the creation of graphical editing environment for applications on the eclipse platform. Main components of the GEF are the Draw2D⁴ libraries and the Standard Widget Toolkit (SWT) (Northover and Wilson, 2004).

GEF is designed in a manner so that the user will be able to create graphical representation of models, which are generated by EMF, by providing data

¹Eclipse project - Eclipse Modeling Framework, at: <http://www.eclipse.org/modeling/emf/>, last accessed (11th October, 2012).

²EMF/FAQ, What is EMF? EMF Wiki Category, at: <http://wiki.eclipse.org/EMF/FAQ>, last accessed (11th October, 2012).

³Eclipse project- Graphical Editing Framework, at: <http://www.eclipse.org/gef/>, on-line access (11th October, 2012).

⁴Draw2d, at: <http://www.eclipse.org/gef/draw2d/index.php>, last accessed (11th October, 2012).

into the components of a software design pattern known as Model-View-Controller. The model consists of the user defined data (i.e. the EMF models), the view consists of figures (i.e., the graphical icons which will be used to represent the EMF models) and the controllers consists of libraries that modify the models and update the views whenever the user makes changes to the EMF models and/or their graphical representation. Hence, once an EMF model is modified the effected view is refreshed by the controller (Rath and Varro, 2006).

As shown in Figure 5.2, each "Model" is associated with a specific "Figure" that graphically represents the "Model" element. The mapping between the "Model" element and the "Figure" is achieved with the "EditPart" classes of the GEF. Normally, each "Model" element has its corresponding "EditParts" class, which has the role of the controller in to the Model-View-Controller software pattern. The "EditParts" classes specify how the "Model" element is mapped into its visual "Figure", and how the "Figure" will behave to specific types of user interactions.

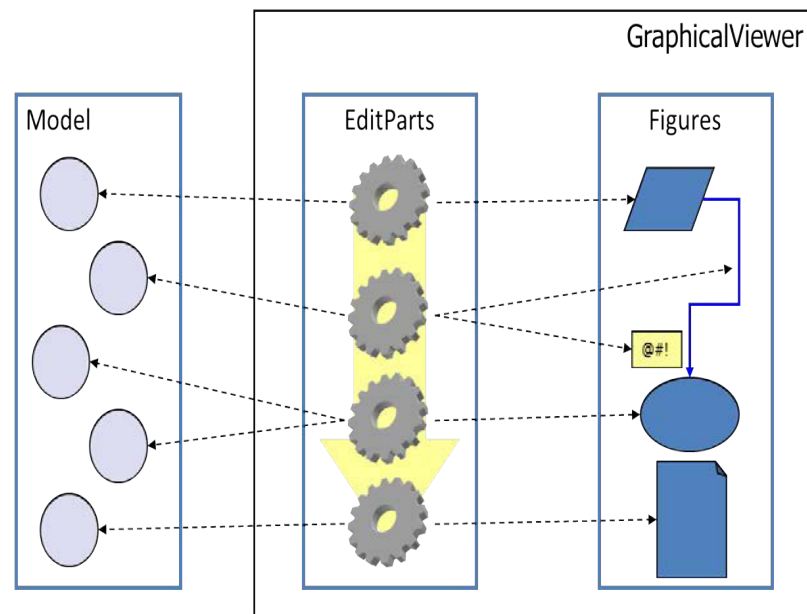


Figure 5.2: The GEF model view controller pattern (Hudson and Shah, 2005).

5.2.3 EVL

EVL (Kolovos et al., 2012) is a language, which allows the developers to apply the constraints and the appropriate validations in to the domain models. The validators check that the user models do not contain any inconsistencies, which may be emerged due to missing or incompatible information. The

constraints enforce some limitations in the creation of the user models. For example, the constraints can be used to ensure that graphical user models, which represent specific concepts that are not should be associated, will not be linked together. The EVL validations are defined at the meta-model level M2, which are later checked at the user model level M0.

5.2.4 EGL

EGL (Kolovos et al., 2012) is a template-based language that can be used to transform models into various types of textual artefacts, including executable code (e.g. Java), reports (e.g. in HTML), images (e.g. using DOT), formal specifications (e.g. Z notation (Spivey, 1992)), or even entire applications comprising code in multiple languages (e.g. HTML, Javascript and Cascading Style Sheets (CSS) (Kolovos et al., 2012)). In EGL, text based artefacts are produced by combining static text information from the templates and the information from the user models.

5.2.5 XML, Java and XPath

XML is a data description language, which is commonly used for storing and exchanging data across multiple systems. It can also be used to facilitate the generation and management of meta-data and allows the creation of many vocabularies, as standards for various domains. An XML document is composed of XML elements with a start-tag "<" and an end-tag ">". The information between the tags, if any, is called content of an element, where each element can have additional information described by its attributes. The tags in an XML document indicate the data meaning and not the data appearance (Zisman, 2000) .

Java is an object oriented programming language where programs are built from classes. A number of objects can be created from a defined class. These objects then are called instances of the class. A class contains fields and methods. Fields are data variables belonging either to the class itself or to the objects of the class. Methods are collections of statements that operate on the fields to manipulate their data by assigning, for example, values to fields and other variables, evaluating arithmetic expressions, invoking methods, and controlling the flow of execution.

XML Path Language (XPath) is a language for selecting information from XML documents. It treats XML as a tree of various node types such as an element node, an attribute node and a text node. To locate these nodes from

a tree structured XML document, path expressions are used. An XPath expression is a series of location steps separated by slashes "/". In each step a set of nodes in relation to the current node is selected. These nodes become the current node(s) for the next step. The set of nodes, which are selected by the expression, are the nodes remaining after processing each step in order. A location step consists of: a) an axis b) a node test and c) zero or more predicates. The axis specify the tree relationships between the nodes selected by the location step and the current node (e.g., ancestor, ancestor-or-self, attribute, child, descendant, descendant-or-self); a node test is used to identify a node within an axis, by specifying a node type or the node name (e.g. text(), node()); a predicate is an expression placed inside square brackets, used to further refine the set of nodes selected by the location step (e.g. [@Type='IT']) (Carminati et al., 2005).

5.2.6 PostgreSQL and JDBC

PostgreSQL is an open source object-relational database management system through its use of SQL as a query language. It provides an easy-to-use graphical interface for managing and developing PostgreSQL database and to perform complex queries on the stored data. The Java DataBase Connectivity (JDBC) driver is a set of classes and methods available in Java programming language to establish a connection with a database from the computer program.

5.3 Summary

This chapter described the software platform, the modeling frameworks as well as the software technologies used for the development of the hazard and early warning analysis DSML and its dedicated editor. The main criterion for selecting these tools and technologies was their open source nature as well as their efficient support by the development community. These technologies and tools had to be configured and used in order to define the appropriate architecture of the hazard and early warning analysis DSML. Details on the DSML constructs and on the result from the utilisation of these technologies will be given in the following chapters.

*Everything should be made as simple as possible, but
not simpler.*

Albert Einstein



Hazard and Early Warning Analysis DSML: Requirements, Specifications and Architecture

THE previous chapters described the alternative strategies for developing a DSML as well as the rational for selecting the "development from scratch" strategy for the hazard and early warning analysis DSML. This chapter will proclaim the requirements and specifications of the DSML and it will describe how they were defined. Furthermore, the architecture of the DSML will be presented. As mentioned in Chapter 3, there are two main tasks that should be accomplished when developing a DSML from scratch. The first is to define the specifications of the DSMLs abstract syntax and the second is to define its concrete syntax. Sections 6.1 to 6.2 will describe in detail the process for defining the abstract syntax of the hazard and early warning analysis DSML and sections 6.3 to 6.3.3 will describe the process for defining its concrete syntax. In essence, this chapter reveals the linguistic characteristics of the hazard and early warning analysis DSML.

6.1 Defining the Meta-models

The steps taken in order to define the meta-models of the hazard and early warning analysis DSML were the following:

- 1) Identify the tasks that should be accomplished when the hazard and early warning analysis is applied.
- 2) List the concepts and relations that should be represented into the DSML in order to make the creation of user models, during the execution of each task of the analysis, feasible.

- 3) Define the requirements, based on which the meta-models will be specified, so that, each user model will "hold" the features and data necessary for the completion of each task of the analysis.

These steps will be described in detail in the following sections.

6.1.1 Hazard and Early Warning Analysis Task Identification

As shown in Table 2.1 the hazard and early warning analysis starts always with the definition of a hazard. Then, the analysts should accomplish the following tasks:

- Represent the real world elements, which comprise the reference system (i.e. create a control structure diagram), and define the feedback control process(es) responsible for keeping the system in a safe state.
- Define the conditions that may lead to accidents (i.e. define the inadequate control actions) based on the feedback control processes which identified in the previous task.
- Define the flaws and the early warning signs associated with each feedback control process.

Table 6.1: The main tasks and subtasks of the hazard and early warning analysis after hazard identification step.

Main Tasks	Subtasks	Sequence No. in Table 2.1
TASK 1	1 -Define the control structure diagram (i.e. the elements responsible for maintaining the safe state of the reference system) (step 1 of STPA); 2 -Define the feedback control process(es), which are "included" in the control structure diagram (step 2a of STPA).	2
TASK 2	3 -For each feedback control process define the inadequate control actions (step 2b of STPA).	3 and 8
	4 -Define and update the process models (i.e. the safety related rules) of the controllers in the feedback control processes (steps 3a of STPA and 3 of EWaSAP).	4
TASK 3	5 -Define the flaws in each feedback control process using the guidewords of STPA which are depicted in Figure 2.3 (step 3b and 4 of STPA).	5
	6 -Specify the warning signs associated to each flaw (step 1 of EWaSAP).	6
	7 -Specify the attributes of each warning sign (step 2 of EWaSAP).	7

As Table 6.1 shows, the main tasks of the analysis are comprised of sub-tasks. The subtasks are deriving by the logic sequence of STPA and EWaSAP steps that, as it was mentioned in section 2.5, compose the hazard and early warning analysis.

6.1.2 Hazard and Early Warning Analysis Concepts and Semantics

The concepts associated to each main task of the analysis together with their semantics are shown in Tables 6.2, 6.3, 6.4.

Table 6.2: Domain concepts associated with first main task and their meanings.

Main Task 1	Concept	Meanings
	Control Structure Diagram	It depicts, in an abstract manner, the elements of a system, which their combined objective is to keep the system in a safe state.
	Feedback Control Process	It is a process with four functions, namely that of Measure, Compare, Compute, and Correct. These functions are performed by elements of the reference system, which take the roles of the sensor, the controller and the actuator. The process is applied over a controlled process to make sure that it will not deviate by its desired state.
	Controlled Process	Represents a process within the system that is monitored and controlled by the elements of the feedback control process.
	Sensor	Represents the real world element responsible for “measuring” some aspects of the controlled process.
	Controller	Represents the elements of the feedback control process whose responsibility is to compare the data received by the sensors and compute the appropriate corrective actions that should be enforced into a controlled process.
	Actuator	It is a role that is assigned to those elements of the feedback control process who are responsible for the execution of “corrective” actions over a controlled process.
	Hierarchical level	It is a value that is used to express the hierarchical level in which the element of the feedback control process belongs to.

Table 6.3: Domain concepts associated with second main task and their meanings

	Concept	Meanings
Main Task 2	Inadequate Control Action	Express the inadequacy of a corrective action, which is enforced by the controller via the actuator to the controlled process. There are four general categories of inadequate control actions as step 2b of STPA mentions (See section 2.2) : <ol style="list-style-type: none"> 1. A required control action to maintain safety is not provided. 2. An incorrect or unsafe control action is provided that induces a loss. 3. A potentially correct or adequate control action is provided too early, too late, or out of sequence. 4. A correct control action is stopped too soon.
	Process model	Expresses the set of rules, or the logic, possessed by a controller based on which the most appropriate control actions are selected upon receipt of sensor data.

Table 6.4: Domain concepts associated with third main task and their meanings

	Concept	Meanings
Main Task 3	Causal Factor	Expresses the flaws in a feedback control process that may lead to the realization of a system level hazard. As described in section 2.2 STPA provides a set of guidewords, or generic flaws, that are listed below: <ol style="list-style-type: none"> 1. Incorrect or no information provided by the sensor; 2. Measurement inaccuracies by the sensor; 3. Feedback delays by the sensor; 4. Inadequate or missing feedback by the sensor; 5. Inappropriate ineffective or missing control action by the controller; 6. Inadequate control algorithm by the controller; 7. Process model inconsistent incomplete or incorrect by controller; 8. Control output or external information wrong or missing by controller; 9. Inadequate operation by actuator; 10. Delayed operation by actuator; 11. Process input missing or wrong by controlled process; 12. Unidentified or out of range disturbance by controlled process; 13. Component failures by controlled process; 14. Changes over time by controlled process; 15. Process output contributes to system hazard by controlled process.
	Early warning sign	A definition of this concept is given in section 2.1.1. It is the value of an observation, or of a series of observations, made by the sensor element, which according to the mental models possessed by the controller indicates the presence of causal factors to a potential loss.
	Attributes of an early warning sign	It refers to the characteristics of early warning signs. As described in section 2.4, these are the Source, Means of data transfer, Receiver and Timestamp.

6.1.3 DSML Meta-model Requirements

After comprehending the tasks and the concepts of the hazard and early warning analysis, the next step in defining the DSML meta-model specifications was to collect, with the guidance of domain experts, those features of the analysis, which should be specified into the meta-model of the DSML. This step is necessary in order to help the intended users¹ of the DSML to create models that will represent exactly what they need during the execution of each task and subtask of the hazard and early warning analysis.

Two sets of requirements were identified during this step. The first set

¹As noted in chapter 1, the intended users of the DSML are the professionals in particular domain (i.e. safety analysts).

comprises of the requirements, which apply when users executing more than one main tasks of the domain. Table 6.5 depicts these requirements. The second set comprises of the requirements, which apply when users executing just one main task. Tables 6.6, 6.7, 6.8 and 6.5 depicts these requirements.

For example, the user of the DSML must have in his disposal a representation of the concept "inadequate control action" (i.e. Table 6.7), when attempting to create models for the 2nd main task of the domain. This concept, however, is not necessary for the creation of models that represent the results of the 1st or 3rd task of the analysis. Therefore, defining the concept of "inadequate control action" into the meta-model of the DSML is a requirement that belongs in to the second set of requirements. To the contrary, the requirement of a "name" property in each concept of the DSML (i.e. Table 6.5), is common to all tasks of the domain. Thus, the provision of this attribute to the concepts and relations of the DSML is a requirement that belongs in to the first set of the requirements.

Table 6.5: Requirements necessary for the execution of at least two main tasks of the domain

Main Tasks	Common Requirements
1, 2 and 3	1. The user should be able to provide a name and a description to all concepts represented by the DSML.
1 and 2	2. The user should be able to represent the concepts of "sensor", "controller", "actuator" and "controlled process" of a feedback control process in his models.

Table 6.6: Requirements necessary for the execution of a first main task of the analysis

Main Task 1	Specific Requirements
	<ol style="list-style-type: none"> 1. The user should be able to define more than one control structure diagrams. 2. For each control structure diagram the user should be able to define its <ol style="list-style-type: none"> a. Hierarchical level of a control structure in the reference system. b. Potential role (i.e. controller, sensor, actuator, control process) in relation to an analysis, which started from a higher hierarchical level. 3. The control structure diagram can contain elements of the feedback control processes of the system. 4. The user should be able to define the role (i.e. controller, sensor, actuator, controlled process), of an element contained in a control structure diagram that belongs in to a feedback control process of the system.

Table 6.7: Requirements necessary for the execution of a second main task of the analysis

	Specific Requirements
Main Task 2	<ol style="list-style-type: none"> 1. The user should be able to assign inadequate control actions into the controller element and not into the controlled action, sensor, and actuator elements. 2. The user should be able to assign more than one inadequate control actions to each controller. 3. The user should be able to provide a description to each control action. 4. The user should assign only one out of the four types of inadequacy that characterises each inadequate control action. 5. When a user creates a controller in his models he must able to specify a process model (i.e. mental model) with its basic characteristics such as its inputs, output and rules. 6. The user should be able to specify the goals and/or intentions of each controller. 7. The user should be able to specify the inputs and outputs of each controlled process. 8. The user should be able to connect a sensor element only with a controlled process and a controller element. 9. The user should be able to connect a controller element only with a sensor and an actuator element. 10. The user should be able to connect an actuator element with a controller and a controlled process element. 11. The user should be able to connect a controlled process element only with an actuator and sensor element.

Table 6.8: Requirements necessary for the execution of a third main task of the analysis

	Specific Requirements
Main Task 3	<ol style="list-style-type: none"> 1. The user should be able to specify the causal factors of each hazard (i.e. to model a possible hazard analysis scenario). 2. When a user defines a hazard he should be able to provide information about its effects as well as an abbreviation code. 3. When a user defines a causal factor to a hazard he should be able to select the most appropriated STPA guideword as its description. 4. A user should be free to assign more than one early warning signs to each causal factor. 5. The user should be able to specify the properties of an early warning sign, which are the source, means of data transfer, receiver and warning sign (see section 2.4). 6. The user should not be free to connect a hazard element with itself or with another hazard element, but only to connect a hazard to a causal factor. 7. The user should be able to connect the causal factors with other causal factors.

6.2 Meta-Model Specifications

Having produced the list of requirements of the meta-model, the next step was to transform the requirements in to specifications using the Ecore meta-modeling language. As mentioned in section 3.3.1.2 the basic constructs of the Ecore are the EClass, EAttribute and EDataType and EReference. In addition to these basic concepts, the EEnum concept was used for the specification of the meta-model of the hazard and early warning analysis DSML. EEnum enables the definition of attributes into meta-model elements that their values, during the user model creation, can be selected from a predefined list. In the GMF meta-modeling tool these Ecore constructs are represented by the graphical symbols that are shown in Table 6.9.

Table 6.9: Ecore concepts and their graphical representation in GMF






Ecore Concepts	GMF Icons
EClass	
EAttribute	
EDatatype	
EReference	
EEnum	

Figure 6.1 shows an example of the Ecore meta-model using the GMF meta-modeling tool. In this example the concept of "Controller" has been specified in the meta-model of the DSML using the EClass element. The "Controller" EClass inherits the attributes of the "NamedElement", "Process-Model" and "ControllerElement" EClasses as per requirements five and six of Task two in Table 6.7 and requirement one in Table 6.5. The notion of inheritance in the Ecore is depicted in Figure 6.1 with the symbol "->".

Furthermore, the "Inadequate_Control_Actions" EReference element is assigned to the "Controller" EClass to allow the users of the DSML to selectively assign one or many "inadequate control action" elements in it (i.e. note that the concept of the inadequate control action in Figure 6.1 is defined with an EEnum) as per requirement four of Task two in Table 6.7.

In addition, the requirements that form the constraints of the DSML are represented with an UML association. For example, the requirement eight and eleven of Task 2 in Table 6.7, where the user should be able to connect a controlled process only with a sensor or an actuator element has been represented in Figure 6.1 with the "ContProc_TO_Sensor_Link" and with the "Actuator_TO_ContProc_Link" EClasses. The Ecore meta-models for each task of the analysis have been created in a manner similar to the above example.

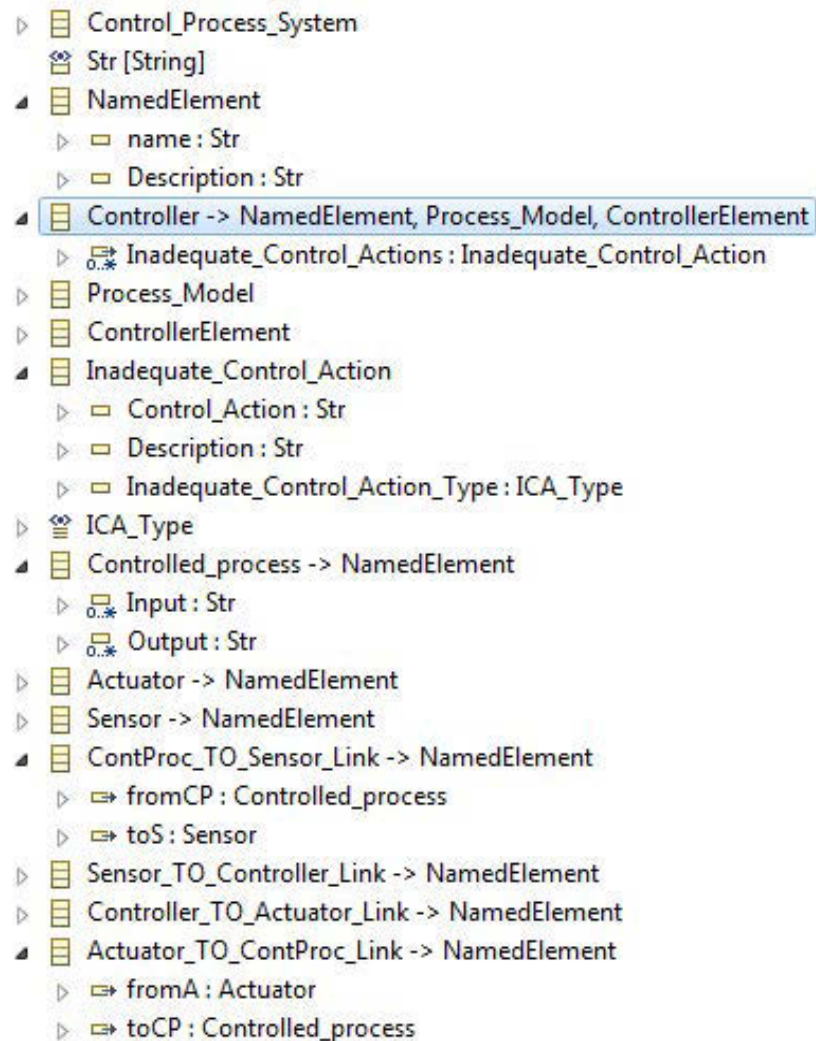


Figure 6.1: Ecore meta-model specifications.

Figures 6.2, 6.3 and 6.4 represent the meta-model specification of each main task of the hazard and early warning analysis using UML annotations. The UML class represent an EClass. The UML generalization represents the inheritance of the EClass and the UML association represent the EReference of the Ecore models. The "0..*" UML relations represent some constraints, like for example in Figure 6.2 where it is used to expresses that the user should be able to create "more than one" "Control structure diagram" as in requirement 1 of task 1 in the Table 6.6. The Ecore model also dictates, in a sense, which concepts will require a concrete syntax so that to be displayed to the users. For example, the concept of "control structure diagram" will be graphically represented by its appropriate concrete syntax, whereas concepts, such as "hierarchical level" and "role", are designed as properties of this concept and do not require graphical representation of their own. In practice, the domain experts where the ones who selected the constructs of the DSML

that needed graphical icons, having as criterion, the elements that must be "dragged and dropped" from a pallet in to a canvas when creating their user models.

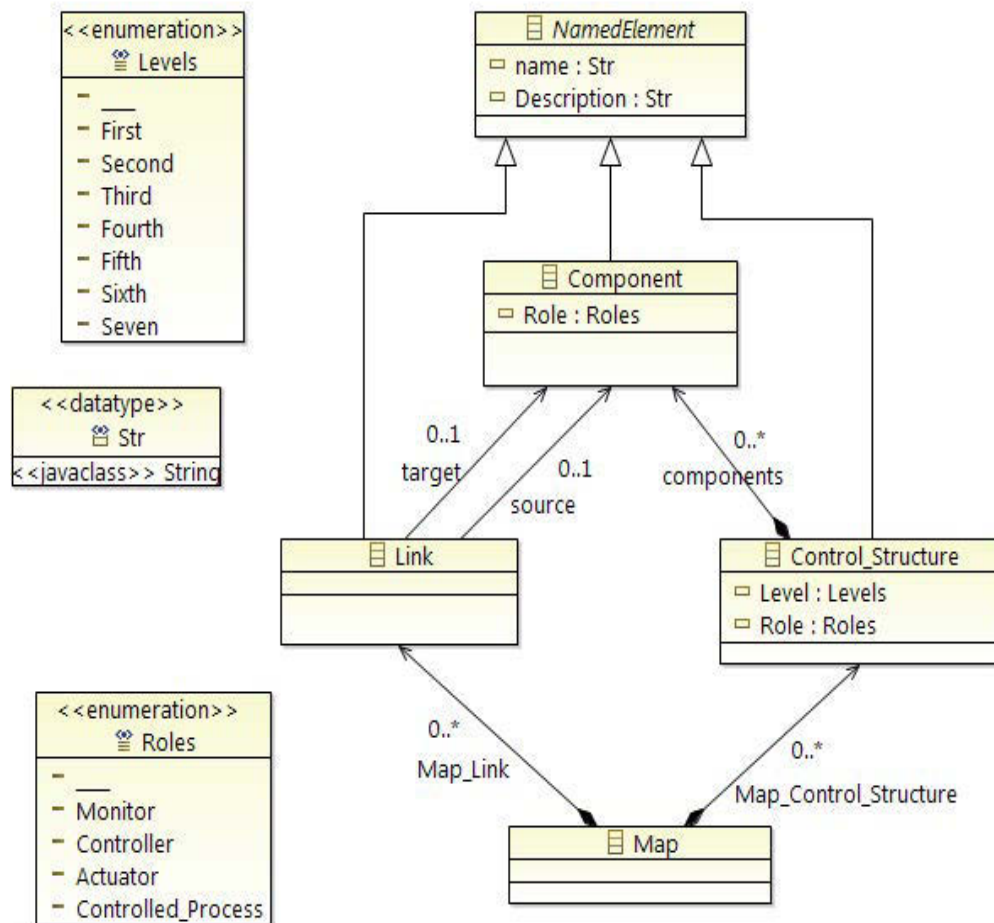


Figure 6.2: DSML meta-model specific for the first main task.

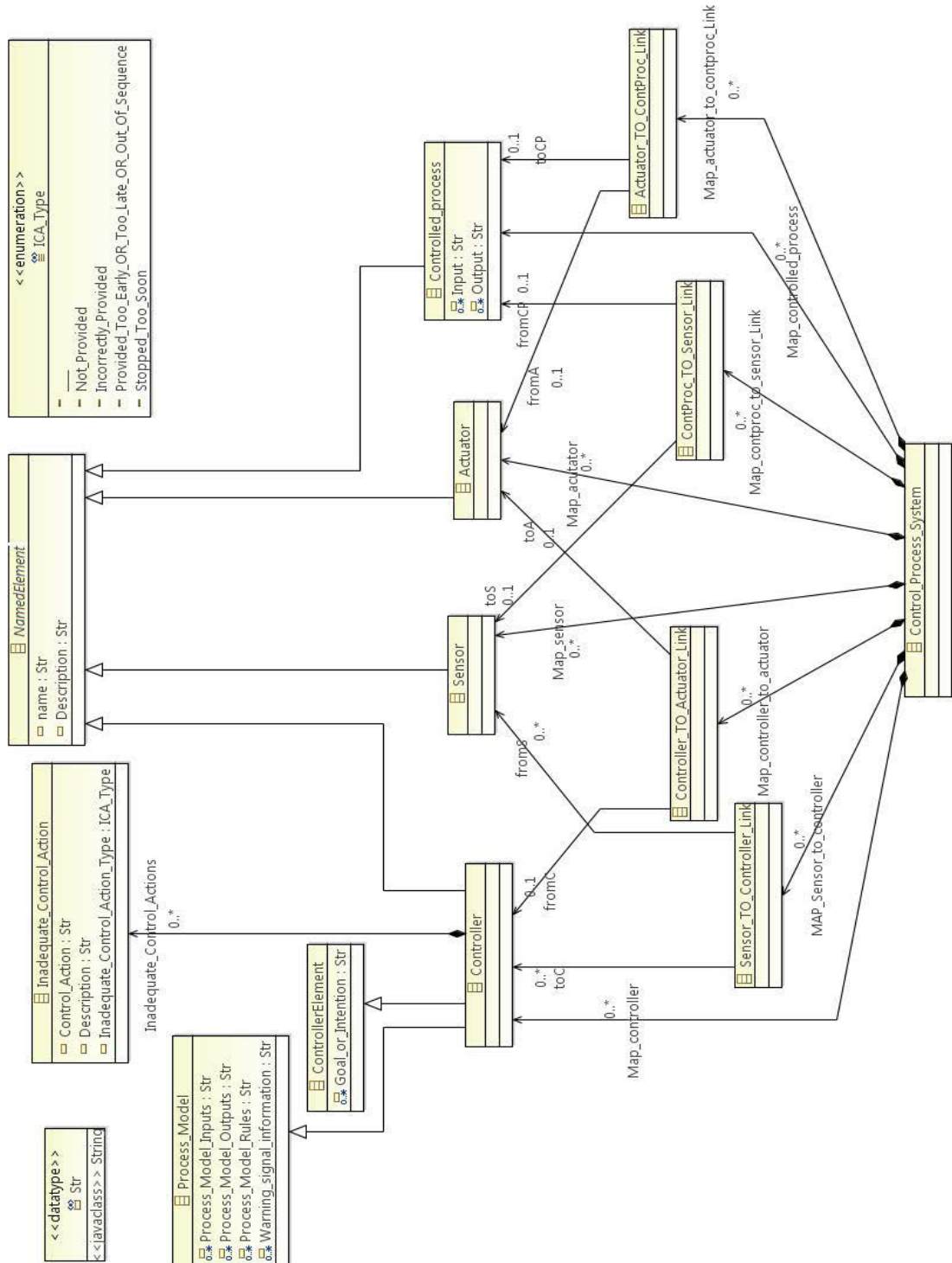


Figure 6.3: DSML meta-model specific for the second main task.

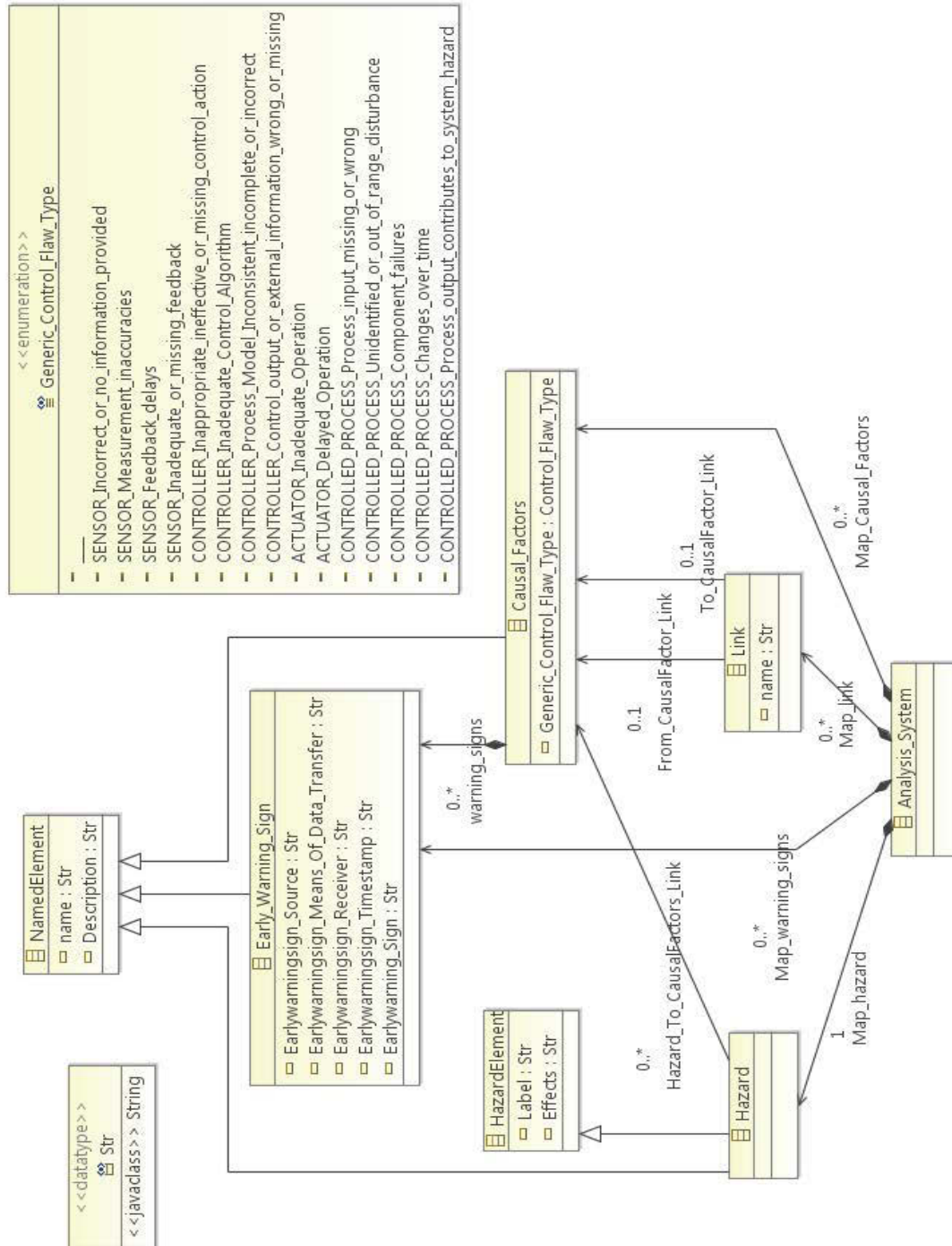


Figure 6.4: DSML meta-model specific for the third main task.

6.3 Concrete Syntax Specifications

As mentioned at the beginning of this chapter, after the meta-model specification step the next step is to define the concrete syntax of the meta-model elements. In order to do this the following steps must be accomplished.

- 1) Identify the graphical icons that will represent the meta-model elements to the users of the DSML;
- 2) Based on the results of the previous step, specify the concrete syntax of the DSML;
- 3) Define the appropriate and necessary validation checks on the user inputs.

These, steps will be described in the following sections.

6.3.1 Selection of Graphical Icons

The following procedure was followed to identify the most appropriate set of graphical icons, to represent the elements of the meta-model elements that are in need of concrete syntax.

For each element a set of two graphical icons alternatives were shown into a panel of three domain experts and were asked to select for each concept that icon, which was most intuitive based on their perceptions to the concept of the domain. That process was repeated with different icons four times. The icons, which prevailed in votes during each repetition of the selection process, were kept aside in order to be used for the formation of four distinct sets of icon alternatives (i.e. each new icon set was not the same with any of those set of icons which were prevailed in the selection process). These sets were formed in two groups of two sets of icons and then each group was shown again to the panel of domain experts in order to select the most appropriate icon for each concept. This selection process produced the final two sets of icons, which were shuffled first and then presented for selection for the last time to the panel of experts. Tables 6.10 , 6.11 and 6.12 shows the details of the final selection process where the fifth column in each table indicates the prevailing icon based on the number of votes given (i.e. see column number six) by the domain experts of the panel.

Table 6.10: Selection of graphical icons for the first main task










	Concept	Set A Graphic icons	Set B Graphic icons	Selected Icon	No of votes
Main Task 1	Control Structure diagram				2
	Component				3
	Link				3

Table 6.11: Selection of graphical icons for the second main task











































	Concept	Set A Graphic icons	Set B Graphic icons	Selected Icon	No of votes
Main Task 2	Inadequate Control Action				2
	Controller				3
	Actuator				2
	Controlled process				2
	Sensor				2
	Sensor_TO_Controller_link				2
	Controller_TO_Actuator_Link				2
	Actuator_TO_ContProc_Link				2
	ContProc_TO_Sensor_Link				2

Table 6.12: Selection of graphical icons for the third main task

Main Task 3	Concept	Set A Graphic icons	Set B Graphic icons	Selected Icon	No of votes
	Hazard				3
	Causal Factor				3
	Early Warning sign				2
	Hazard_To_CausalFactors_Link				3
	Link				2

6.3.2 Concrete Syntax Specifications

After selecting the appropriate graphical icons of the meta-models the next goal was to specify their concrete syntax with EAnnotations, which is an add-on to Ecore meta-model tool. Figure 6.5 shows how EAnnotation constructs were used to specify the concrete syntax of the meta-model element "Controller". For example, as shown in Figure 6.5, the EAnnotation construct "label" has the value "name" in order to display the name, which a user may give to a "Controller", whereas the "label.icon" construct has the value "true" define that the "Controller" will have a specific graphical icon as concrete syntax.

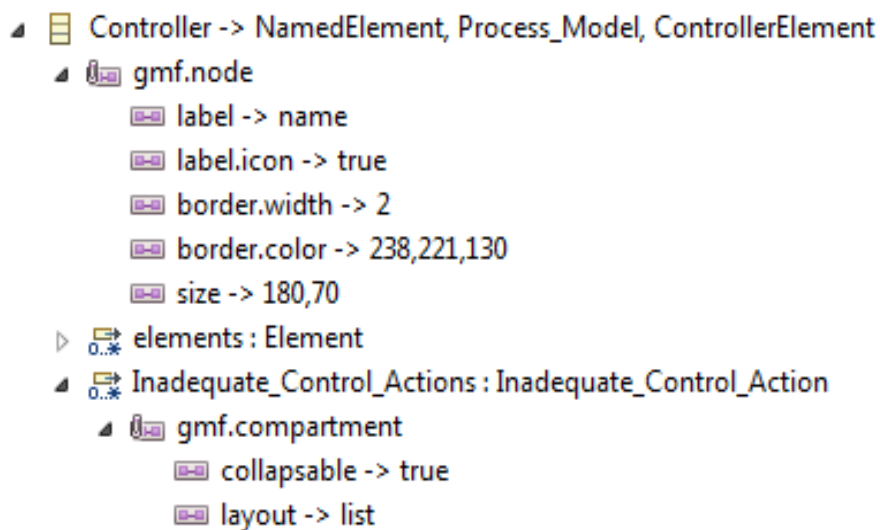


Figure 6.5: Snippet of annotations specified for meta-model of ICAML.

6.3.3 Validations

The last step in defining the concrete syntax of the meta-model is to define the validations for a set of user inputs. The purpose of doing this is to add a mechanism in to the DSML, which will inform or restrict or correct the users whenever they enter wrong or illegal datatype inputs. Table 6.13 shows the validations over the DSML on the user inputs. These validations were originally devised by the designer and then enriched by the domain experts. Each validator may be enforced to many concepts during the creation of user models. Tables 6.14, 6.15 and 6.16 show, which validations from the Table 6.13 were applied in each domain concept.

Table 6.13: Validations over user models

Validators	Validation Code
Check that the user has entered an input in to the “name” attribute	A
Validate that the value of the “name” attribute begins with capital letter	B
Ensure that the value of the attribute is not left blank	C
Ensure that the default value of the attribute is displayed	D
Check that the input value of an attribute consist of alphabetic or numeric characters	E
Ensure that the character length of the input value does not exceed the twenty characters	F

Table 6.14: Enforced model validations on the domain concepts of the first main task

Main Tasks 1	Concept	Attributes	Validations (See Table 6.13)
	Control Structure diagram	Name	A, B, C, E, F
		Description	E
	Component	Name	A, B, C, E, F
		Description	E

Table 6.15: Enforced model validations on the domain concepts of the second main task

Main Task 2	Concept	Attributes	Validations (See Table 6.13)
	Controller	Name	A, B, C, D, F
		Description	E
	Process Model	Inputs	E
		Outputs	E
		Rules	C, E
		Warning Signal information	E
	Actuator	Name	A, B, C, D, F
		Description	E
	Controlled process	Name	A, B, C, D, F
		Description	E
		Inputs	C, E
		Outputs	C, E
	Sensor	Name	A, B, C, D, F
		Description	E
	Inadequate Control Action	Control Action	C, E
		Description	E

Table 6.16: Enforced model validations on the domain concepts of the third main task

Main Task 3	Concept	Attributes	Validations (See Table 6.13)
	Hazard	Name	A, B, C, E, F
		Description	E
		Label	E, F
		Effects	E
	Causal Factor	Name	A, B, C, E, F
		Description	E
	Early Warning Sign	Source	C, E
		Means of data transfer	C, E
		timestamp	C, E
		Receiver	C, E
		Warning Sign	C, E

After identifying the validation requirements of the DSML the objective was to specify them using the EVL language. EVL comprises of three main constructs: (1) the "Context" where the validations will be applied; (2) the "check" where the specific code for the validation is specified (3) the "message", which defines what will be displayed as a message when the check part will not validate the code. Figure 6.6 depicts an example of EVL code. Assuming that a user has not entered a name for a "Component" in his model, the EVL code of the example will activate a message, as Figure 6.7 shows, which will display "Please enter a name" so that to inform the user for his omission.

```

context Component {
  constraint HasName {
    check : self.name.isDefined()
    message : 'Please enter the name: Unnamed ' +
              self.eClass().name + ' not allowed'
  }
}

```

Figure 6.6: Snippet of EVL code.

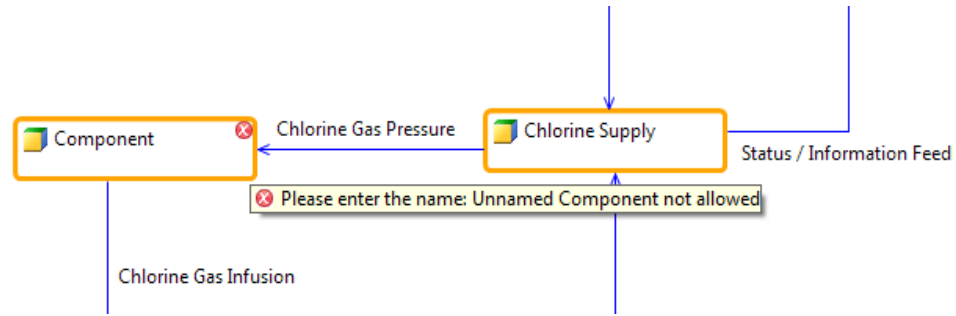


Figure 6.7: Validation on user model.

6.4 DSML Architecture

Having the abstract and concrete syntax of the DSML defined, the point has been reached, where the architecture of the entire DSML was ready to be finalised. The main issue at this point was to decide if one or more meta-models should be defined in the M2 level of the architecture, resulting in to the creation of one or more DSMLs.

As it is shown by other works in DSML, more than one DSMLs can be defined and then integrated to create a solution in a problem domain. In these cases, the number of the DSMLs was dictated by the number of the main tasks, which according to the domain experts, must be accomplished. For example, in Genie (Bencomo, Grace, Flores, Hughes and Blair, 2008b) the domain had two main tasks in need of modeling namely, the environment or context variability, and the structural variability. Therefore, two DSMLs were integrated in one solution. That accords also with (Van Deursen et al., 2000), (Zdun, 2002) and (Lewis and Launchbury, 1998) that advocates that domain specific language should be small and specific to the tasks of the domain.

In the case of the hazard and early warning analysis DSML, the domain has three main tasks. Some tasks, share the same meta-model elements. For example, the controller, sensor and actuator elements are shared between the first and second tasks. However, in the context of the first task the purpose of their existence is to represent the physical elements of the reference system, whereas in the second task their purpose is to help the users of the DSML to define the flaws of the feedback control processes. There is, in essence, a different "methodological" need that must be satisfied with these concepts in each task of the domain and therefore the elements are shared. Thus, it was decided to define three meta-models into level M2 of the architecture of the hazard and early warning analysis DSML, each one specific to the main tasks of analysis.

As a result, there will not be one hazard and early warning analysis DSML. Instead, there will be three task oriented DSMLs that will be integrated together in one prototype software editor in order to allow the intended users to create models of the hazard and early warning analysis tasks in a consecutive manner. One DSML will represent the concepts of the first task of the hazard and early warning analysis. This DSML is called Control Structure Modeling Language (CSML) and the UML representation of its meta-model is depicted in Figure 6.2, the second DSML is called Inadequate Control Action Modeling Language (ICAML) and its UML meta-model representations is depicted in Figure 6.3 and the third DSML is called Analysis Modeling Language (AML) and its meta-model representation is depicted in Figure 6.4.

The final architecture of the proposed solution is shown in Figure 6.8. the M2 level contains the abstract syntax of the three DSMLs. Level M1 contains the concrete syntax of these DSMLs. In level M0 the three DSMLs are integrated in a software editor that allows the intended users to create hazard and warning analysis models. Finally, the model transformation layer contains the generated computer code from the user models.

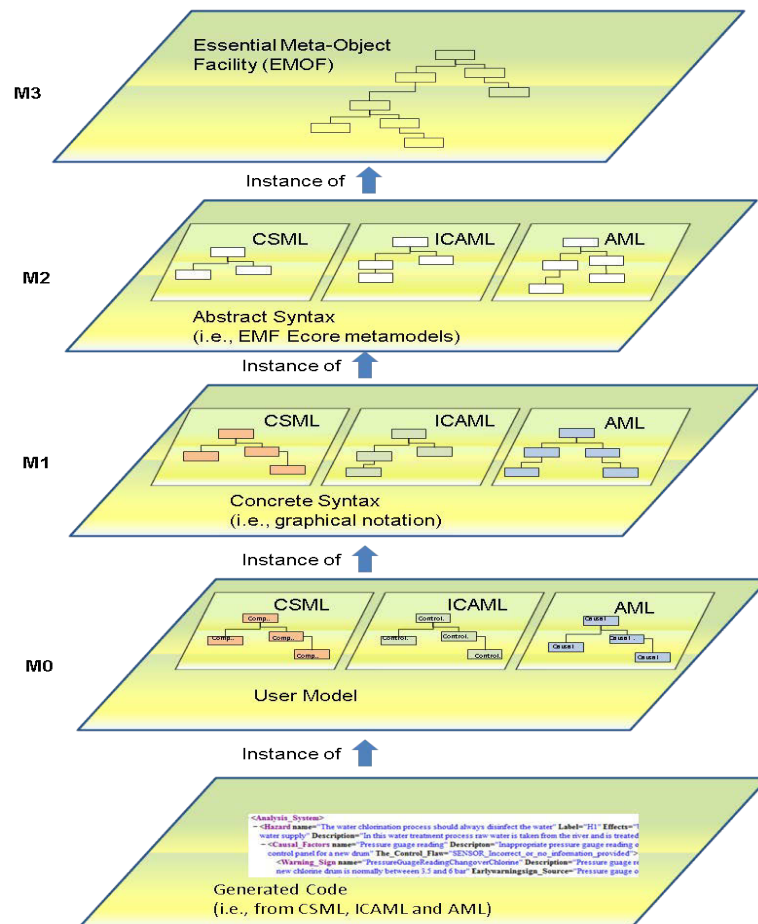


Figure 6.8: The architecture of the proposed solution comprised by three DSMLs.

6.5 Summary

This chapter described the process based on which the abstract and concrete syntax of the three early warning sign analysis DSMLs have been defined. At the beginning of this process the author, with the guidance of domain experts, has identified the tasks of the analysis and has listed their associated concepts and attributes. The result of this phase was to group the tasks of the domain into three main tasks. In the second stage of the process, the author defined the abstract syntax of the DSML. He then defined the concrete syntax of those concepts, which should have graphical icons. The graphical icons were selected by a panel of domain experts.

In addition, the architecture based on which the three DSMLs were designed in order to enhance the creation process of hazard and early warning analysis models has been presented. The architecture contains three meta-models at the level M2, namely the CSML, ICAML, and AML meta-models, each one of which is dedicated to a separate task of the domain. The purpose of the CSML is to enable the users to define the boundaries of the reference system and its key elements. The purpose of the ICAML is to enable users to create models that represent the feedback control processes of the reference system and to define their inadequate control actions. The purpose of the AML is to allow the users to create models of accident scenarios and to define the early warning signs at each causal factor to a loss.

The level M1 of the architecture contains the concrete syntax of each meta-model of level M2. Level M0 contains the user models of these three DSMLs and the model transformation layer of the architecture contains the programming code generated by the user models. In level M1 the three DSMLs of the early warning sign analysis are integrated in a prototype software editor. The architecture of this editor, together with its mechanism for the generation of computer code from the user models, will be described in the next chapter.

Good visual layout shows the logical structure of a program.

Steve McConnell

7

Design and Architecture of the Hazard and Early Warning Analysis Editor

THE previous chapter described the requirements and specifications of the three DSMLs, which combined, enhance the creation of hazard and early warning analysis models. This chapter focuses on the design of a prototype software editor, which was created to facilitate the creation of user models, from the constructs of the abstract syntax of the three DSMLs. The goal of this editor is to allow safety practitioners or other professionals, working in critical infrastructures, to conduct hazard and early warning analysis and to generate software code as a result of their analysis. The way in which the DSM technologies were used and combined to create the hazard and early warning analysis editor is not claimed as contribution in this dissertation. However, because the hazard and early warning analysis editor was used extensively to collect the data necessary to prove the hypothesis of this dissertation the description of its design and architecture deemed necessary.

7.1 Prototype Software Editor Requirements

The design of the prototype software editor started with the identification of its requirements. There were two types of requirements: functional and nonfunctional. Functional requirements describe what the system should do. According to Sommerville (Sommerville, 2011), the functional requirements are "statements of services the system should provide, how the system

should react to particular inputs and how the system should behave in particular situations". The non-functional requirements, are not directly concerned with the specific services delivered by the system to its users, however, they usually specify the characteristics of the system as a whole, such as performance, availability, usability (Sommerville, 2011). The functional requirements of the hazard and early warning analysis editor are shown in Table 7.1.

Table 7.1: Functional requirements hazard of the prototype editor

Functional Requirements
<ol style="list-style-type: none"> 1. The user should be able to create models using the appropriate DSML (as Figure 7.1 shows) for each consecutive task and subtask of the early warning sign analysis. 2. A dedicated editor view should be provided for each DSML. 3. A palette must be placed in each dedicated editor to contain the graphical icons representing the abstract syntax of each DSML. 4. A property view must be placed in each dedicated editor to edit the properties on the user models. 5. The users should be able to input, edit and delete information on the specified user models. 6. The editor must have a mechanism in order to prevent user from entering information which refers to same concepts in different dedicated editors. 7. The user should be able to generate code from the user models after finishing his analysis. 8. The user should be able to query and browse the information from the user models.

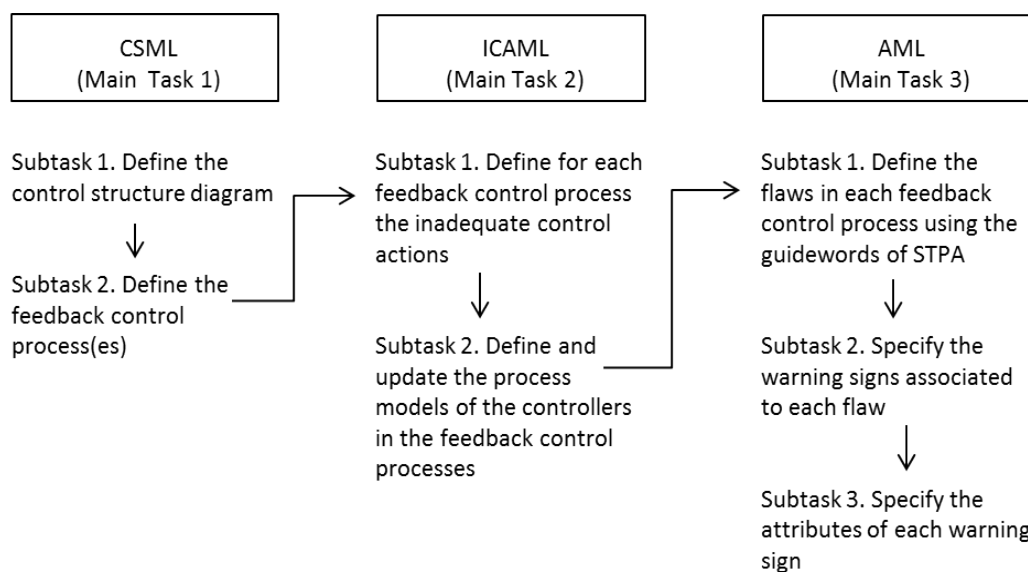


Figure 7.1: The consecutive tasks and subtasks of the domain and the three DSMLs.

A number of non-functional requirements have been reported for software products, such as robustness, usability, security and reliability requirements. From the users' perspective, the most important non-functional requirement is in this case the usability. The (ISO, 1998) defines usability as: The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use. Table 7.2 shows the usability requirements of the hazard and early warning analysis prototype editor.

Table 7.2: Usability requirements of the hazard and early warning analysis prototype editor.

Usability Requirements
1. Effectiveness: The user should be able to complete all the tasks of the early warning sign analysis.
2. Efficiency: It should be less time consuming for the users to complete the task of the analysis using the editor compared to the manual completion of each task.
3. Satisfaction: The editor should have an easy to use interface and its users should be satisfied by its use.

7.2 Architecture

The architecture of the prototype hazard and early warning analysis editor is shown in Figure 7.2. It has a four layer architecture consisting of :

- 1) Dedicated editors' layer;
- 2) Ecore models layer;
- 3) Code generation layer;
- 4) Database layer.

The first layer contains the editors with the appropriate graphical interfaces that a) enable the creation of user models and b) enables its users to query and extract user model data. The Ecore models, which define the abstract syntax of each DSML reside in the second layer. Whereas, the code generator facilities of each DSML resides into the 3rd layer of the architecture. The 4th layer contains the database facility, which is responsible for the storage of the generated code.

Each layer of the architecture is described in detail in the following sections.

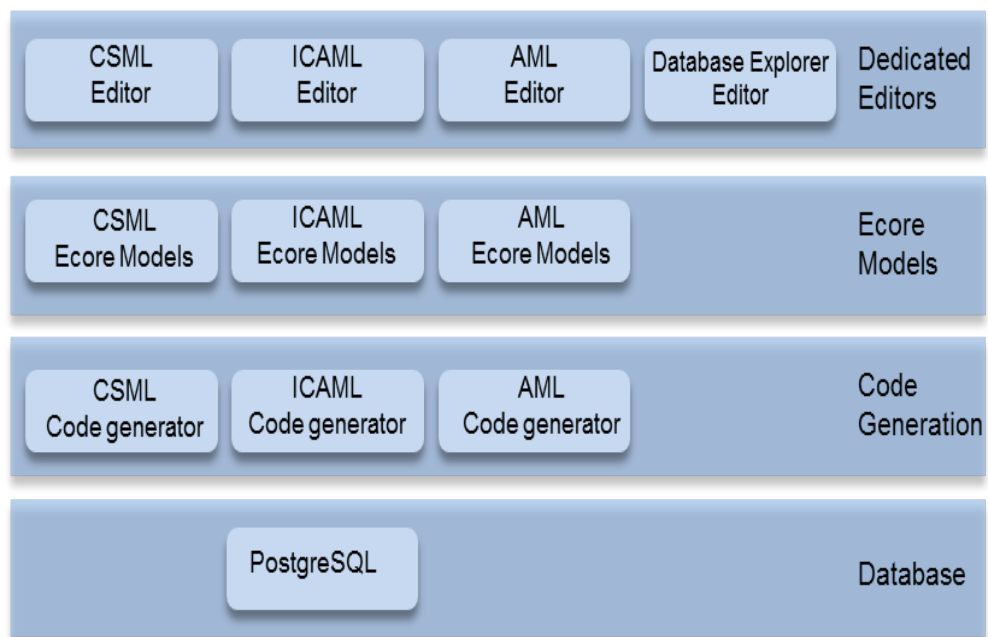


Figure 7.2: The architecture of the prototype hazard and early warning analysis editor.

7.2.1 Dedicated Editors Layer

The Dedicated Editors Layer is the top layer of architecture and comprises of four editors. The *CSML Editor*, *ICAML Editor*, *AML Editor* and *Database Explorer Editor*. The first three editors are GMF based editors and each of them corresponds to a domain specific modeling language (i.e. CSML, ICAML, and AML described in section 6.4) and the fourth editor is a database explorer editor that enables the users to execute queries over the data stored in the database.

The process based on which the three GMF based editors are created from their Ecore models during runtime is shown in Figure 7.3. From a domain model (e.g. CSML Ecore), the corresponding graphical definition model (i.e. .gmfgraph) and tooling definition model (i.e. .gmftool) and the mapping definition model (i.e. .gmfmap) are generated.

The tooling definition model provides a "blueprint" of the layout, which the dedicated DSML editors will have. As result to this, there are some features that are commonly available in the editors, such as the commands "open", "save", "print", "copy", "paste and "undo" and the features such as menu bar, buttons, dropdowns, windows and other controls to give commands and enter data in each active editor.

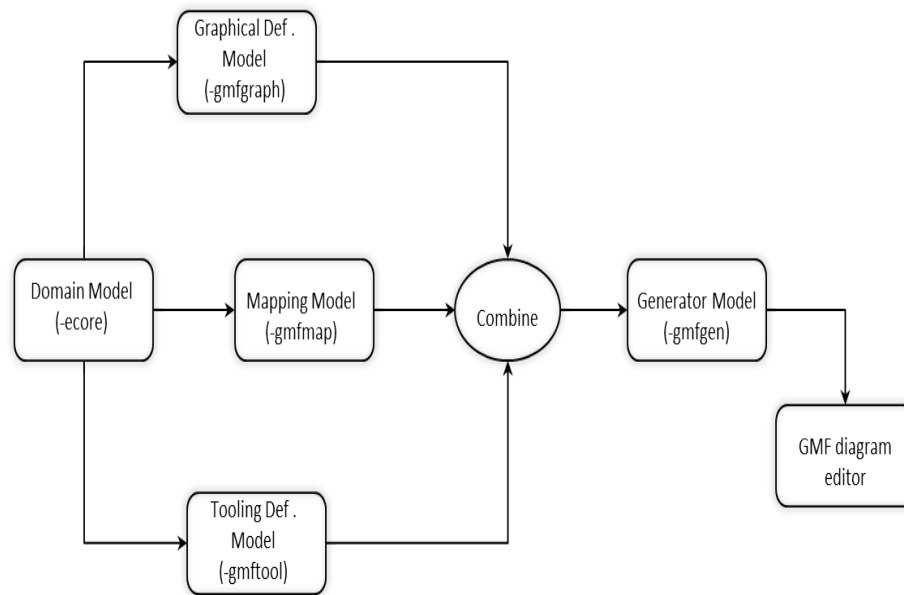


Figure 7.3: Generation of a GMF based editor from the domain model.

One additional common feature provided by the tooling definition is the three elements that form the layout of each DSML editor. These are : 1) *Palette*; 2) *Editor View* and 3) *Property View*. The *Palette* is the placeholder for a set of graphical icons representing the concepts of a domain specific modeling language, such as components and links. The support of drag-and-drop feature on these editors allow users to perform "grabbing and dragging", that is, to select a graphical icon from the palette and dropping it onto the editor view. The editor view represents the main window of the editor, where the user model diagrams are drawn. The property view is used to display the properties or attributes of the graphical icons placed on the editor view of an editor and allows user to insert, edit and delete the information on them. The graphical definition model provides the concrete syntax of each DSML needed for the definition of the graphical elements that will be displayed in the editor. The mapping definition model serves for establishing a link between the graphical definition model and the tooling definition model by creating the generator definition model (i.e. .gmfgen). The generator definition model is then used to generate the GMF editor.

Furthermore, each dedicated editor has a code generation button, as Figure 7.4 shows, that enable users to generate software code from their models. This button is created by integrating Eclipse extension point (i.e. org.eclipse

.ui.actionSets¹(Gamma and Beck, 2004)) functionality in to the GMF dedicated editors.

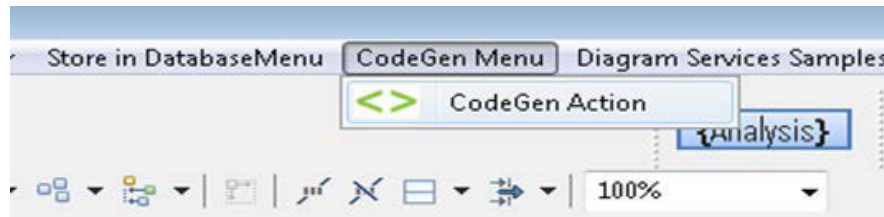


Figure 7.4: Graphical icon to generate code on the menu bar.

The database query editor was originally developed by open source project named Eclipse SQL Explorer² and it was integrated as plug-in to the prototype software editor. In order to enable queries over the user models there was a need to store the information of the models in to a database. That was achieved at first with Hibernate (Bauer and King, 2006). However, the users were not always providing values into the "id" property of the elements of their models. As consequence it was not possible to execute queries in every case because the "id" value was needed to identify each element of the user models and their relations. Therefore, it was decided to remove the "id" property from the meta-model specifications of the DSMLs and to generate xml documents that describe the user models. This document then was used to retrieve the answers to the queries. This approach required to parse the XML document and to insert an "id" value at the runtime to each tag of the xml document that corresponds to the elements of the user models as shown in the snippet of Figure 7.5. Then, the parsed XML document is saved into the relational database by opening a JDBC connection.

Altogether three database schemas were created to store the information from the use of each hazard and early warning analysis DSML. Figure 7.6 shows the database schema with its referential integrity constraints developed to store the information of the models created using the constructs of AML.

¹Further information available at http://help.eclipse.org/indigo/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2Fworkbench_basicext_actionSets.htm, last accessed (11th October, 2012).

²Eclipse SQL Explorer, at <http://www.sqlexplorer.org/index.php>, last accessed (11th October, 2012).

```

212 private ArrayList getParentNodes(Element hazardElement)
213 {
214     NodeList nl_HazardhasChildNodes = hazardElement.getChildNodes();
215     int countParentNode = 1;
216     int HazardhasChildNodes = nl_HazardhasChildNodes.getLength();
217     Node node = null;
218     ArrayList parentNodesArrayList = new ArrayList();
219     Element el_parentId = null;
220
221     if (HazardhasChildNodes > 0)
222         for (int khk = 0; khk < HazardhasChildNodes; khk++)
223         {
224             node = nl_HazardhasChildNodes.item(khk);
225             if (node.getNodeType() == Node.ELEMENT_NODE)
226             {
227                 Element el_parent = (Element) node;
228                 for (int khkh = 0; khkh < el_parent.getAttributes().getLength(); khkh++)
229                 {
230                     if (el_parent.getAttributes().item(khkh).getNodeName().contentEquals(Control_Process))
231                     {
232                         String parentNodeName = el_parent.getAttributes().item(khkh).getNodeValue().toString();
233                         int id_num = countParentNode;
234                         el_parentId = setId(el_parent, id_num); // ## Call setId() method
235                         countParentNode++;
236                     }
237                     if (el_parentId.getAttributes().getNamedItem(ID).getTextContent().toString() != null)
238                     {
239                         String parentNode_Id = el_parentId.getAttributes().getNamedItem(ID).getTextContent().toString();
240                     }
241                 }
242                 parentNodesArrayList.add(el_parentId);
243             }
244         }
245     return parentNodesArrayList;
246 }
247
248 private Element setId(Element el, int Id) {
249     int numOfAttributesInNode_before = el.getAttributes().getLength();
250     String ID = "Id";

```

Figure 7.5: Snippet of the java program used to insert an "id" value.

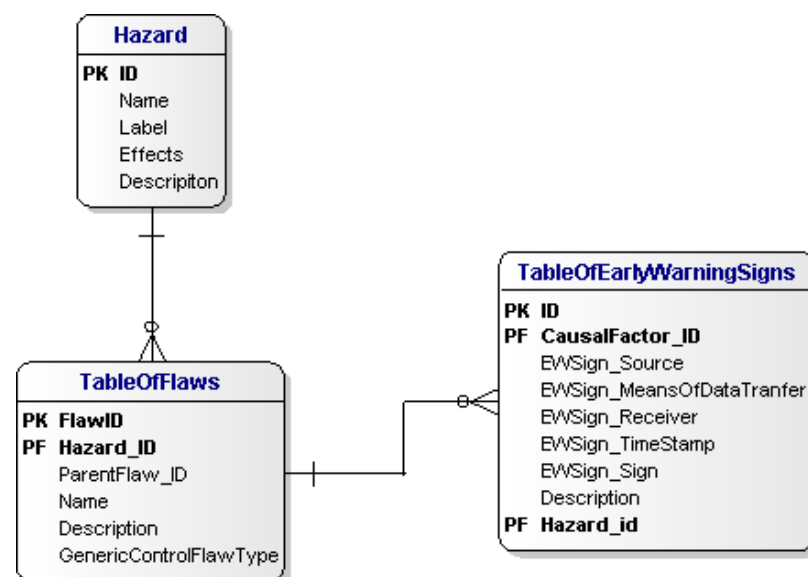


Figure 7.6: Database schema with its referential integrity constraints .

7.2.2 Ecore Model Layer

The second layer of the architecture is comprised by the three Ecore models, which are CSML Ecore Model, ICAML Ecore Model and AML Ecore Model. As mentioned in section 3.3.1.2. The Ecore meta-models contain the specifications on all concepts and relationships that must be depicted in the respective dedicated editors (i.e. CSML editor, ICAML editor, and AML editor). Furthermore, this layer contains the model validations, which were described in detail in section 6.3.3.

7.2.3 The Code Generation Layer

This is the third layer of the architecture and comprises mainly of three code generators, which are the CSML Code Generator, ICAML Code Generator and AML Code Generator. The code generators are used to generate the code from the user models specified on their respective editors (i.e. CSML editor, ICAML editor, and AML editor). The code generators are template based M2T transformations and are executed when the user press the code generation button shown in Figure 7.4.

The code generator mechanisms were defined with EGL (Kolovos et al., 2012). EGL has two main elements the static and the dynamic section. The contents of the static section do not change during the code generation process and appear directly in the generated code. The contents of the dynamic section change, similarly to a variable, which its output value in the generated code is equal to the text that the user entered in to his models.

For example, referring to Figure 7.7, the tag pair [% %] is used to delimit a dynamic section as shown in lines 11 to 17 . Any text not enclosed in such a tag pair is contained in a static section as shown in lines 21 to 25. The [%=expr %] construct, appends "expr" to the output code generated by the transformation as "[%=Hazard.name%]" shown in line 6. Also, static and dynamic sections are used together, as it is shown in lines 6 to 9 or 27 to 29 of Figure 7.7.

Executing this EGL template will produce the generated JAVA code which is shown in Figure 7.8. The EGL template that was used to generate the XML code is shown in Figure 7.9 and that code which is generated by this is shown in Figure 7.10.

```

3 public class Analysis {
4   [% if (Hazard.isDefined()) { %]
5   class hazard {
6     String name = "[%=Hazard.name %]";
7     String Lbl = "[%=Hazard.Label %]";
8     String Eff = "[%=Hazard.Effect %]";
9     String Desc = "[%=Hazard.Description %]";
10  }
11  [%} %]
12  [% for (Causal_Factors in Hazard_To_CausalFactors_Link.allInstances()) { %]
13  [% if (Causal_Factors.isDefined()) { %]
14  [% var i : Integer = i+1; %]
15  [% i = i+1; %]
16  [% var Causalfactor : String; %]
17  [% Causalfactor.concat(i); %]
18
19  class "[%=Causalfactor %]" {
20
21    hazard h = new hazard();
22    String Hazard_name = h.name;
23    String Hazard_label = h.Lbl;
24    String Hazard_Effect = h.Eff;
25    String Hazard_Description = h.Desc;
26
27    String CausalFactor_Flaw = "[%=Causal_Factors.The_Control_Flaw %]";
28    String CausalFactor_Name = "[%=Causal_Factors.name %]";
29    String CausalFactor_Description = "[%=Causal_Factors.Description %]";
30    [% for (EarlyWarning_Sign in Causal_Factors) { %]
31    String EarlyWaring_sign= "[%=EarlyWarning_Sign.Earlywarning_Sign %]";
32    String EarlyWaringSign_source = "[%=EarlyWarning_Sign.Earlywarningsign_Source %]";
33    String EarlyWaringSign_means_of_data_transfer = "[%=EarlyWarning_Sign.Earlywarningsign_Mean %]";
34    String EarlyWaringSign_receiver = "[%=EarlyWarning_Sign.Earlywarningsign_Receiver %]";
35    String EarlyWaringSign_timestamp = "[%=EarlyWarning_Sign.Earlywarningsign_Timestamp %]";
36    [%} %]
37  }
38  [%} %]
39  [%} %]
40 }

```

Figure 7.7: Snippet of EGL template for generating Java code.

```

5
6 public class Analysis{
7
8 @class hazard {
9     String name = "The water disinfection process should always disinfect the water";
10    String Lbl = "H1";
11    String Eff = "Untreated drinking water supply";
12    String Desc = "In this water treatment process, raw water is taken from a river and is treated with chlorine";
13 }
14
15 @class Causalfactor1 {
16
17     hazard h = new hazard();
18     String Hazard_name = h.name;
19     String Hazard_label = h.Lbl;
20     String Hazard_Effect = h.Eff;
21     String Hazard_Descripton = h.Desc;
22
23     String CausalFactor_Flaw = "CONTROLLER Inappropriate ineffective_or_missing_control_action";
24     String CausalFactor_Name = "Chlorine gas added to the water";
25     String CausalFactor_Description = "In the disinfection element of water chlorination process";
26
27     String EarlyWaring_sign = "Audible alarm and blinking LED";
28     String EarlyWaringSign_source = "Residual chlorine detector in the disinfection element";
29     String EarlyWaringSign_means_of_data_transfer = "Alarm sound and flashing light";
30     String EarlyWaringSign_receiver = "Water treatment works manager";
31     String EarlyWaringSign_timestamp = "2010122213004";
32 }
33 }

```

Figure 7.8: Generated Java code.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 [% if (Analysis_System.isDefined()) { %]
4 <Analysis_System>
5 [% if (Hazard.isDefined()) { %]
6     <Hazard name="[%=Hazard.name %]"
7         Label="[%=Hazard.Label %]"
8         Effects="[%=Hazard.Effect %]"
9         Description="[%=Hazard.Description %]" >
10
11 [% for (Causal_Factors in Hazard.allInstances()) { %]
12     <Causal_Factors name="[%=Causal_Factors.name %]"
13         Description="[%=Causal_Factors.Description %]"
14         The_Flaw="[%=Causal_Factors.The_Control_Flaw %]" >
15 [% for (EarlyWarning_Sign in Causal_Factors) { %]
16     <Warning_Sign source="[%=EarlyWarning_Sign.Earlywarningsign_Source %]"
17         Means_of_data_transfer="[%=EarlyWarning_Sign.Earlywarningsign_Means_Of_Data_Transfer %]"
18         Receiver="[%=EarlyWarning_Sign.Earlywarningsign_Receiver %]"
19         Timestamp="[%=EarlyWarning_Sign.Earlywarningsign_Timestamp %]"
20         Waring_sign="[%=EarlyWarning_Sign.Earlywarning_Sign %]" >
21     </Warning_Sign>
22 [% } %]
23 </Causal_Factors>
24 [% } %]
25 </Hazard>
26 [% } %]
27 </Analysis_System>
28 [% } %]
29

```

Figure 7.9: Snippet of EGL template for generating XML code.

```

-<Analysis_System>
- <Hazard name="The water disinfection process should always disinfect the water" Label="H1" Effects="Untreated
drinking water supply" Description="In this water treatment process, raw water is taken from a river and is treated
with chlorine">
- <Causal_Factors name="Chlorine gas added to the water" Description="In the disinfection element of water
chlorination process" The_Flaw="CONTROLLER_Inappropriate_ineffective_or_missing_control_action">
  <Warning_Sign source="Residual chlorine detector in the disinfection element"
  Means_of_data_transfer="Alarm sound and flashing light" Receiver="Water treatment works manager"
  Timestamp="2010122213004" Waring_sign="Audible alarm and blinking LED"> </Warning_Sign>
  </Causal_Factors>
</Hazard>
</Analysis_System>

```

Figure 7.10: Generated XML code.

7.2.4 The Database Layer

Lastly, the database layer of the architecture contains the PostgreSQL database management system, which is a JDBC compliant database. The snippet in Figure 7.11 shows an example of the code that is executed when a user presses the code generation button shown in Figure 7.4 in order to store the XML code generated by the user models in to the database. The code first ensures that the database is connected, and then the SQL insert statement is executed to store the generated code, which represents the components of the control structure diagram, into the database.

```

24
25 try{
26     if (!dbcon.ConnectDB().isClosed()){
27         stat = dbcon.ConnectDB().createStatement();
28     } catch (SQLException e){
29         e.printStackTrace();
30     }
31     return stat;
32 }
33
34 public void update_Hazard_Table(int Id, String el_name, String el_labl, String el_eff, String el_decr){
35
36
37 public void update_TableOfFlaw(int Id, int el_parent_Id, int el_Haz_id, String el_name, string el_GenericFlawType, String el_decr){
38
39
40 public void update_TableOfEW_Signs(int Id, int el_CFact_Id, int el_Haz_id, String el_ENSource, String el_EWDataTrf, String el_EWRec, String
41
42     String UpdateTableOfEW_Signs ="UPDATE \TableOfEarlyWarningSigns\ SET ID='"+Id+"', " +
43     "CausalFactor_ID='"+el_CFact_Id+"', " +
44     "EWSign_Source='"+el_ENSource+"', " +
45     "EWSign_MeansOfDataTransfer='"+el_EWDataTrf+"', " +
46     "EWSign_Receiver='"+el_EWRec+"', " +
47     "EWSign_TimeStamp='"+el_EWTimeSt+"', " +
48     "EWSign_Sign='"+el_EWSign+"', " +
49     "Description='"+el_descr+"', " +
50     "Hazard_id='"+el_Haz_id+"';
51
52
53     try {
54         get_Statement().executeUpdate(UpdateTableOfEW_Signs); // update table
55     } catch (SQLException e){
56         e.printStackTrace();
57     }
58 }
59

```

Figure 7.11: Snippet of the java program used to store the generated code into database.

7.3 Workflow

Having described the architecture of the prototype software editor, this section will present the workflow of conducting hazard and early warning analysis using this prototype software. The user activities, which can/should be done with the editor, are shown in Figure 7.12.

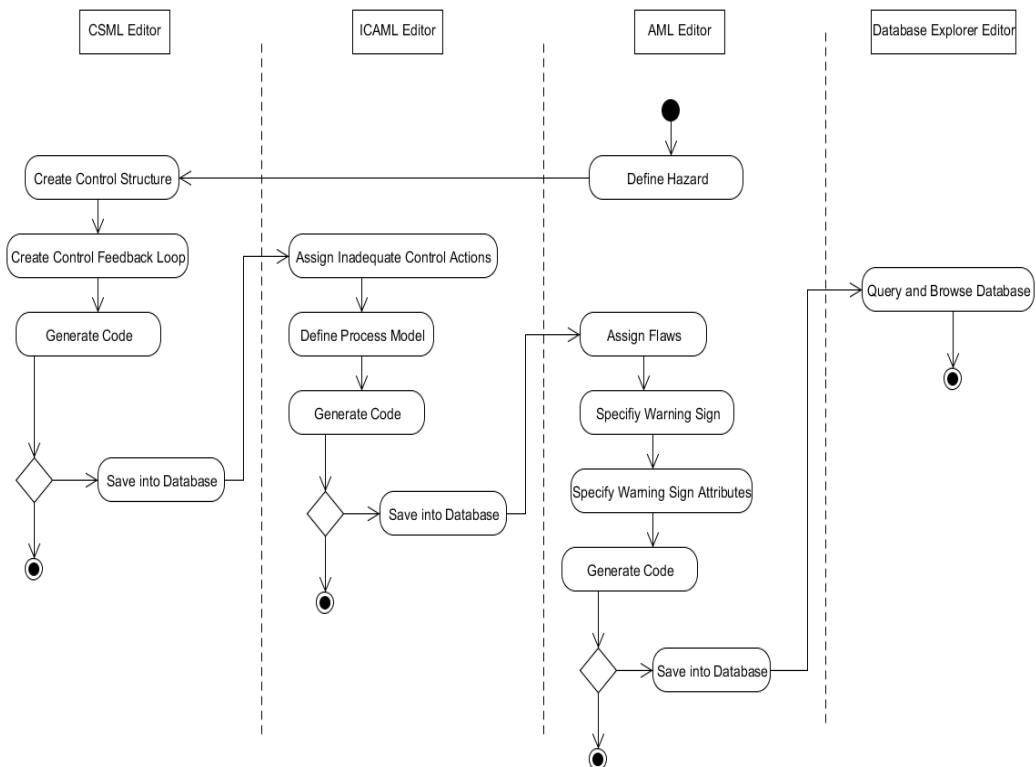


Figure 7.12: UML Activity diagram of prototype early warning sign analysis DSML software tool.

The AML editor is initially used by the user to define hazards that may escalate into accidents. The user then creates the control structure of the reference system with its feedback control processes using the CSML editor. Figure 7.13 shows some elements of the user interface CSML dedicated editor. The palette of the CSML editor contains the graphical icons representing the concepts of "control structure", "component" and "link", which are used to draw control structure diagrams and feedback control processes on its editor view. The attributes of each drawn model are defined and updated using the property view, where dropdown menus enable, in some cases, the users to select an appropriate attribute value.

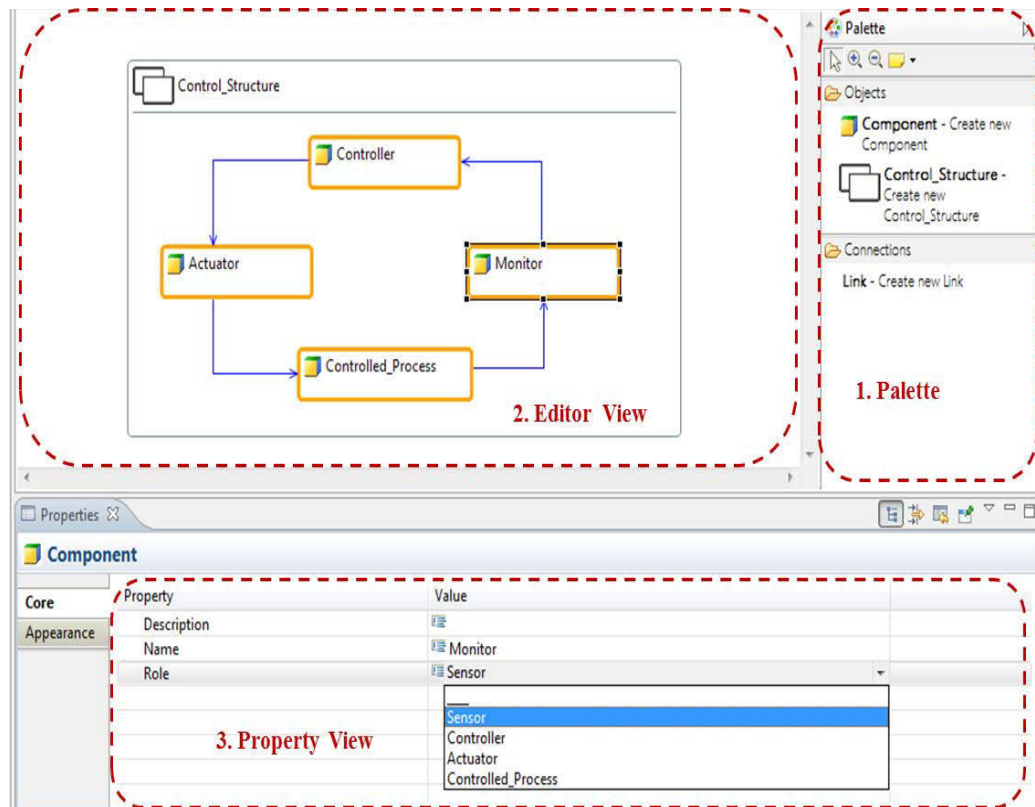


Figure 7.13: The CSML editor.

If the user desires he can save the models drawn on the CSML editor and generate the code and continue his analysis. As the analysis proceeds, the need to assign inadequate control actions to each controller of the feedback control processes and to define the process model for the controller is satisfied by the ICAML editor. Figure 7.14 shows part of the user interface ICAML dedicated editor. The palette of the ICAML editor contains a set of graphical icons representing the concepts of "actuator", "controlled process", "controller", "inadequate control action", "sensor" and "links". These graphical icons are used to draw feedback control processes on the editor view and to assign inadequate control actions to their controllers. The user can select the appropriate type of inadequate control action from a dropdown menu available at the property view. In addition, the process model of each controller of the control structure can be defined and updated by entering data into a text field in the property view.

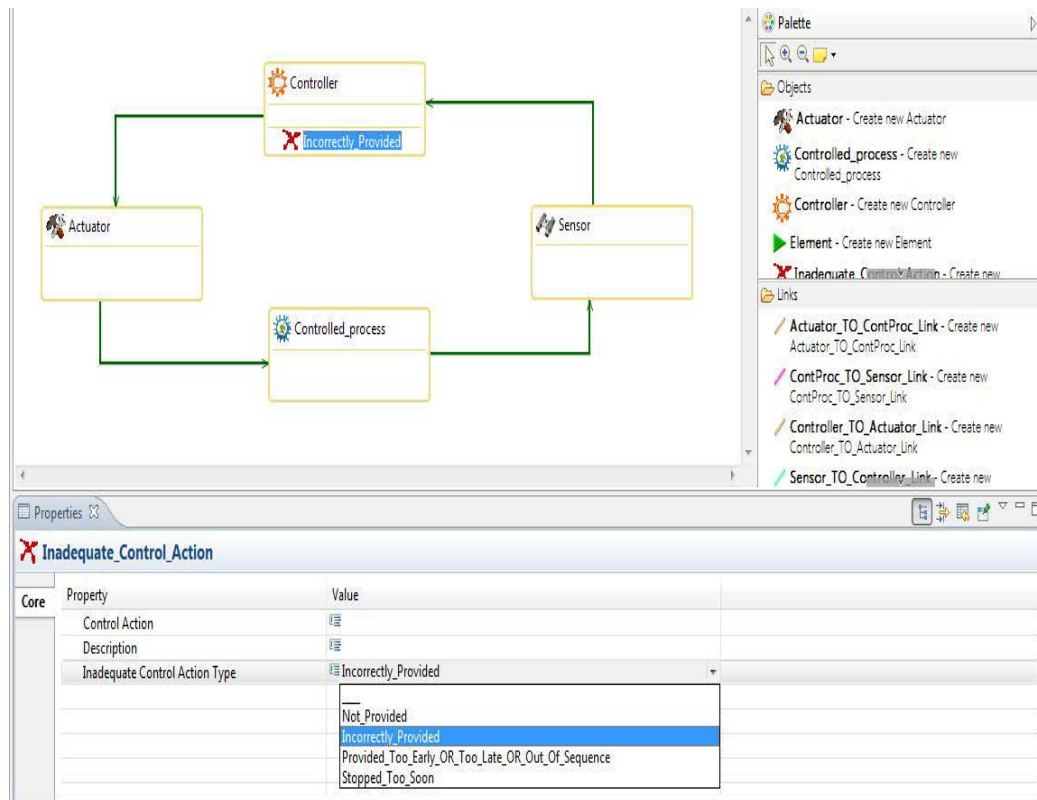


Figure 7.14: The ICAML editor.

Thereafter, the AML editor is used to define the possible flaws and warning signs of a feedback control process. Figure 7.15 shows the user interface of the AML editor. The palette of the AML editor contains a set of graphical icons representing the concepts of hazard, causal factors and links. With these graphical icons, the user can create a hazard analysis scenario where the relations between the hazard and its causal factors are defined. The properties, which are visible from property view, enable users to select the appropriate type of flaw from a drop down menu and to define its associated early warning signs.

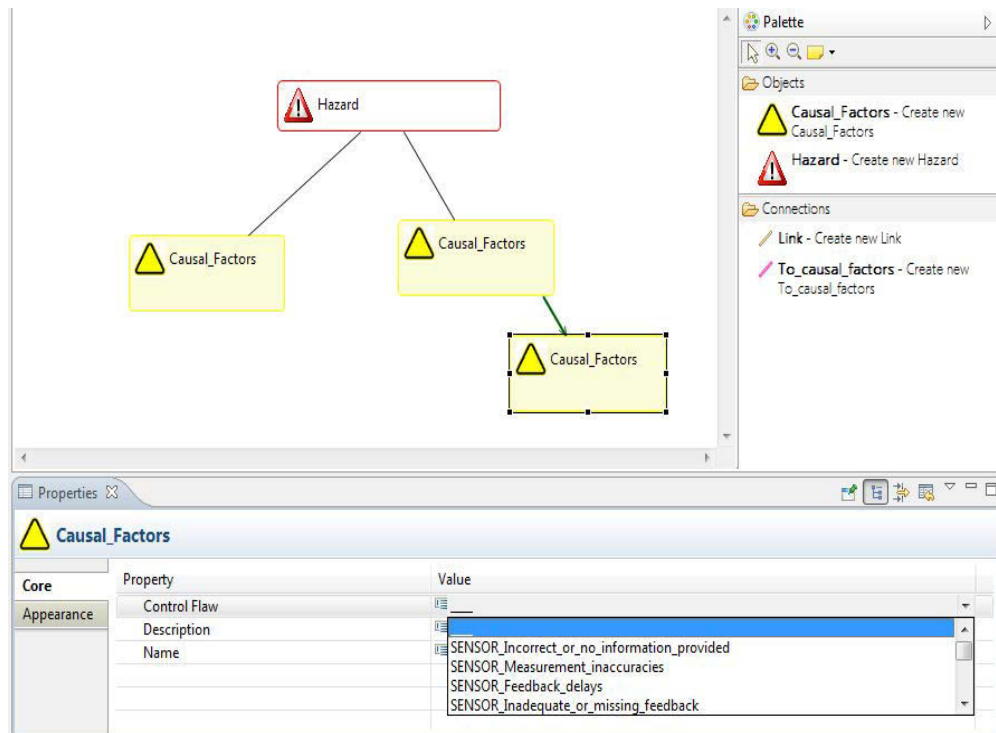


Figure 7.15: The AML editor.

The computer code can be generated from the models drawn in each of the dedicated editors into either in XML or Java format. Furthermore, a database explorer editor enables users to query and browse the user data. As shown in Figure 7.16, the main components of the database explorer editor layout include the: *Connections View* for managing database connections; *SQL Editor View* to execute queries and displays its execution time; *SQL Result View* to display the results on executing Sql statements, *SQL History View* to display successfully executed queries. The information about the number of times the query is executed and the date and time of it execution is also provided in *SQL History View*. The Database Structure View and Database Detail View allow the user to explore the database tables and to view information about the data types of its records.

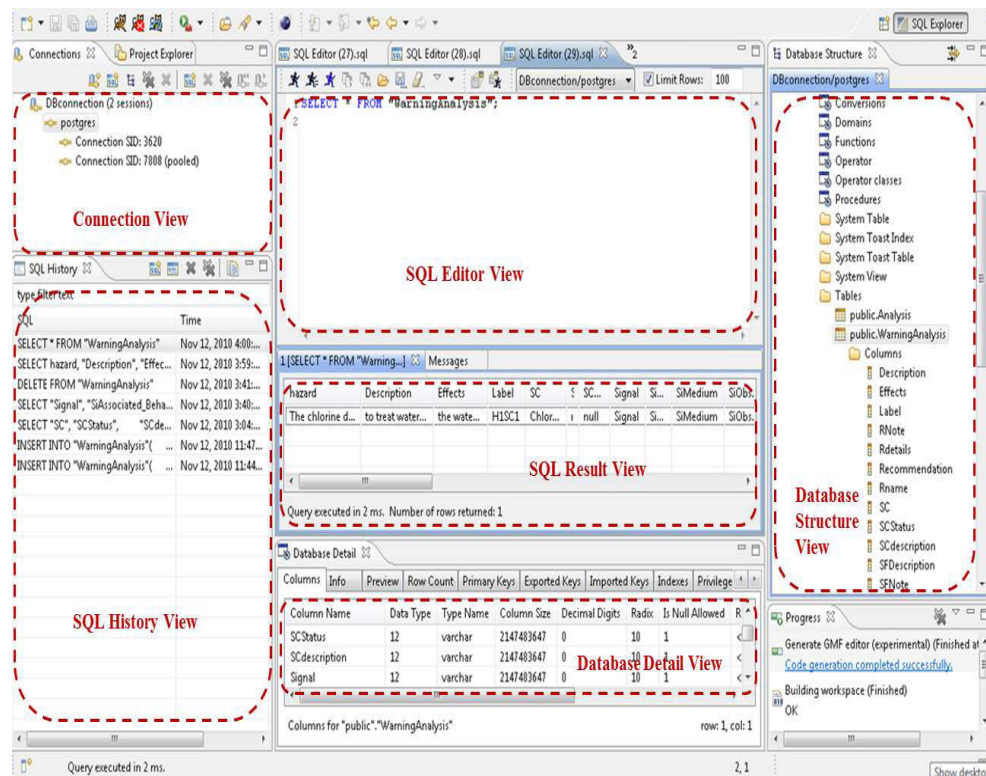


Figure 7.16: The Database Explorer editor.

7.4 Summary

The chapter has presented the functional and non functional requirement of the prototype hazard and early warning analysis DSML editor and provided a detailed description of its architecture.

The architecture consists of four layers. The first layer contains the dedicated editors for the DSMLs described in Chapter 6 and a database explorer editor to enable its users to query and browse the data of their analysis. The second layer contains the Ecore modes and the user model validators for the three DSMLs. The third layer contains three code generators for each DSML. Finally, the forth layer contains the database management system that is used to save and query the generated code from the user models.

Furthermore, the workflow to construct hazard and early warning analysis models using the prototype software editor was described and the layouts of each dedicated editor were presented. A user who would like to run this prototype editor needs to install into his/her computer the Java Runtime environment (JRE)³ and PostgreSQL in order to form the necessary software ecosystem that will make the editor working.

³Information available at <http://www.oracle.com/technetwork/java/javase/downloads/jre-6u25-download-346243.html>, last accessed (11th October, 2012).

Nothing resolves design issues like an implementation.

J. D. Horton

8

Evaluation

HAVING described the architectures of the hazard and early warning analysis DSMLs and of the prototype software editor, this chapter will firstly present the results of a usability evaluation approach of the DSMLs via the use of prototype software editor and secondly, the results of its benchmark tests. The results of the usability evaluation tests were used in order to assess the effectiveness of the DSMLs (i.e. the degree to which the DSMLs allow the users of the prototype software editor to complete all the tasks of the domain) as well as the satisfaction and effectiveness aspects of usability of the prototype software editor. In order to assess the efficiency aspect of usability of the prototype editor benchmark tests were conducted, which measured the time required by the users to conduct the hazard and early warning analysis with and without the prototype editor.

8.1 Overview of the Evaluation Process

The hypothesis in this dissertation is that the three graphical DSMLs described in Chapter 6 enable experts to perform hazard and early warning analysis in a usable manner. Thus, there are two questions that need to be answered.

- 1) *Does the DSMLs enable analysts to complete the tasks of the domain?*
- 2) *Is the prototype software editor presented in Chapter 7 usable?*

In order to answer the first question affirmatively a set of analysts must be able to complete the three main tasks and the seven subtasks of the hazard and early warning analysis, described in section 6.1.1, with the constructs which are provided by the three DSMLs. In order to answer the second question it is necessary to assess, as per the ISO definition of usability that

was mentioned in section 7.1, the effectiveness, efficiency and satisfaction aspects of usability of the prototype software editor. Specifically, as described in Table 7.2, effectiveness implies in this case that the users of the prototype software editor should be able to complete all the tasks of the hazard and early warning analysis. Efficiency implies that it should take less time for the users of the editor to complete the tasks of the analysis compared to the time it takes to complete the analysis manually (i.e. without the use of prototype software editor). Finally, satisfaction implies that the prototype editor should have an easy to use interface that match their expectations.

A response to the first question can be acquired if the assessment on the effectiveness aspect of the usability of the prototype software editor is affirmative. Indeed, the usage of the three DSMLs can be realised only via the prototype editor. Thus, if the usability evaluation results of the prototype editor show that the users were able to complete all the task of the hazard and early warning analysis this will imply that the three DSMLs described in this dissertation are effective.

8.2 Methods

There is a number of software usability models, such as those, which will be described in section 8.2.1. These models extract the user perception on the usability aspects with questionnaires. The focus of the questions in the questionnaire spans all aspects of usability. However, the assessment on the efficiency aspect of the usability based on these models is not so precise. To address this issue, benchmark tests were considered necessary in order to measured the time required by the users of the prototype editor to conduct the hazard and early warning analysis with and without it.

8.2.1 Usability Evaluation Methods

User's perceived usability ratings is a means to assess the usability of a software tool. These ratings are often collected using questionnaires (Vuolle et al., 2008). A number of usability rating methods exist such as the Questionnaire for User Interface Satisfaction (QUIS) (Chin et al., 1988), Software Usability Measurement Inventory (SUMI) (Kirakowski and Corbett, 1993) and System Usability Scale (SUS) (Brooke, 1996).

The QUIS is a questionnaire that was created to gauge the satisfaction aspect of software usability in a standard, reliable, and valid way (Akilli, 2005).

It is arranged in a hierarchical format and contains: (1) a demographic questionnaire, (2) six scales that measure overall reaction ratings of the system, (3) four measures of specific interface factors: screen factors, terminology and system feedback, learning factors, system capabilities, and (4) optional sections to evaluate specific components of the system: technical manuals and on-line help, on-line tutorials, multimedia, Internet access and software installation (Akilli, 2005). Each question is rated on a scale from 1 to 9 with positive adjectives anchoring the right end and negative anchoring the left. In addition, the "not applicable" option is listed as a choice. Additional space, which allows the responder to make comments, is also included within the questionnaire. The comment space is headed by a statement that prompts the responder to comment on each of the specific interface factors. The QUIS 7.0¹ is the current version of the rating method and requires a license for its use.

The SUMI is a questionnaire that provides standardised measurement of user satisfaction with software, which can be used for the evaluation and comparison of products (or versions of a product) and can set and track verifiable targets regarding satisfaction². SUMI measures five aspects of user satisfaction, namely the affect, efficiency, helpfulness, control, and learnability (Bevan, 1995). The questionnaire contains 10 statements for each of the five satisfaction aspects totaling the number of 50, such as "This software respond too slowly to inputs", "I would recommend this software to my colleagues" to which the person who fills the questionnaire has to report if he agrees, disagrees or if he is undecided. The developers of the SUMI recommend administering the questionnaire to a group of 12 participants directly after a test with the product and before any debriefing interview. It can also be used in a survey, with larger groups of users. Similarly to QUIS, SUMI requires a license for its use³.

The SUS is a usability questionnaire consisting of ten standard questions (e.g. "I found the interface was easy to use" , "I think I would not like to use this system frequently"). Each question is accompanied by a Likert's format⁴

¹Information available at <http://lap.umd.edu/quis/>, last accessed (11th October, 2012).

²Information at <http://www.allaboutux.org/sumi?replytocom=2470>, last accessed (11th October, 2012).

³Information available at <http://sumi.ucc.ie/index.html>, last accessed (11th October, 2012).

⁴The Likert scale was named after Rensis Likert, who invented the scale in 1932.

type rating scale, that is a 5 point scale from 1 to 5, with statements ranging from strongly agree, at the left of the scale, to disagree, at the right of the scale. By marking a scale point for each question of the questionnaire the responders indicate their degree of agreement or disagreement. The responders have an opportunity to use the system being evaluated and then fill the questionnaire. The statements are written with positive and negative assertions about the usability aspects of the software and these are listed in the questionnaire alternately. The order of the statements in the questionnaire is numbered. The statements with positive assertions take an even number in the order of the list of statements in the questionnaire whereas the statements with negative assertions take an odd number.

The first step in order to calculate the final SUS score of each questionnaire is to calculate the score of each of its statements. The score of the statements with odd numbers (i.e. 1, 3, 5, 7, and 9) is the number of the user rating number subtracted by one, whereas the score for the statements with even order numbers (i.e. 2, 4, 6, 8, and 10) is five subtracted by the user rating number. The second step is to sum the scores of all statements and multiply the result of the summation with 2.5. The final SUS usability score of each questionnaire ranges from 0 to 100. The higher the SUS score, the higher is the perceived usability. If the SUS score is in the range of 70s it is acceptable, in 80s it is considered good and in 90s it is considered as exceptional (Bangor et al., 2009). SUS was the method which was selected to assess the users' perception upon the usability of the prototype software editor because, as mentioned by (Salvendy, 1997), (Zviran et al., 2006) and (Sauro and Lewis, 2011), it is likely the most popular questionnaire for measuring attitudes toward system usability.

8.2.2 Benchmarking Methods

Benchmarking is the process of testing the performance of a process. There is no single benchmarking process or model. In fact, there are as many benchmarking models as there are authors on the subject (Beatty and Kelley, 1994). However, the benchmarking testing processes have in general similar phases. The preparation phase, where initial tasks and the necessary planning is taking place, the data collection phase, during which the data for assessing the benchmarking measures are collected via tests, the nature of whom depends on what it is to be benchmarked (i.e. a process, a strategy, a software etc.), and finally, the analysis phase where a final result is produced.

There are several forms of benchmarking depending on the objective of

the benchmarking study, such as process benchmarking, strategic benchmarking and performance benchmarking. For example, process benchmarking aims at comparing the methods and practices for performing business processes. It is used to improve the way processes performed every day and its focuses on the day-to-day operations of an organisations (Bogan and English, 1994) (Lankford, 2000). Strategic benchmarking is a comparison of the strategic choices and dispositions made by other companies. It deals mostly with top management and focuses on how companies compete. This form of benchmarking looks at what strategies the organizations are using to make them successful in relation to best practice companies' processes (Bogan and English, 1994) (Lankford, 2000). Performance benchmarking is the comparison of performance measures, such as reliability, quality, speed, time and other product or service characteristics. It is used when organizations want to look at where their product or services are in relation to competitors on the basis of these measures (Bogan and English, 1994) (Lankford, 2000). The approach that was followed in evaluating the prototype software editor is similar to the one recommended by (Shikhar, 2008), for conducting a performance benchmark of a software application. This approach has four main phases: 1) planning; 2) test suite development; 3) performance test execution; 4) analysis and review.

The objective of the planning phase is to identify the number of users involved, the hardware that will be used and the testing approach. At the test suite development phase the testing environment and the performance monitoring process are setup. For example, during this phase some testing models and data population scripts may be created. In the performance test execution phase the tests are executed until the performance testing objectives are met and the results can be obtained. Finally, in analysis and review phase the data collected from the previous phase are consolidated and analysed.

8.3 Evaluation Approach

The evaluation approach of the prototype software editor was conducted in two phases. The first phase aimed at collecting the data about the satisfaction and effectiveness aspect of usability of the prototype editor. The method used during this phase was the SUS questionnaires. For the first phase it was decided to use two SUS questionnaires instead of one. The questions of the first questionnaire dealt with the satisfaction aspect of the usability, whereas

the questions in the second dealt with the effectiveness aspect. This decision was made because of the ten question limit of the SUS questionnaire. More precisely, it was assessed that if one SUS questionnaire was used, with questions which will span all aspects of usability, it will not provide enough data to answer the first question of the evaluation about the effectiveness of the three DSMLs.

The second phase aimed at conducting the benchmarking tests to measure the efficiency aspect of the usability of the prototype editor by measuring the time, a set of experts needed, to conduct the hazard and early warning analysis manually, against the time needed to conduct it with the prototype editor.

The requirements for proceeding with the two phases of the evaluation, were: a) to identify a safety critical system, which will be use as a reference for the hazard and early warning analysis models and b) to identify a set of users/participants.

8.3.1 The Reference System

The safety critical system that selected as reference for the hazard and early warning analysis models was the disinfection process in a drinking water treatment works. The drinking water treatment works is located in Cork City, Republic of Ireland and treats approximately 47,000 meters cubed of water (10 million gallons) per day for a population of 120,000. It is operated and managed by staff 24 hours a day, every day of the year. The water treatment works is managed and administered by a water treatment works manager assisted by a team of operators who follow a series of operating procedures and best practice guidelines to ensure that the works is operating properly and producing clean and wholesome drinking water at all times. In the water treatment process, untreated water (i.e. raw water) is taken from a river (i.e. River Lee) and is treated at the water works to supply drinking water. The various stages or processes of treatment at the water treatment works includes: coagulation, flocculation and clarification followed by filtration, disinfection and fluoridation. In the disinfection process, the water is disinfected using chlorine gas before being pumped into the distribution network.

The disinfection process in water treatment is the most important step to ensure that the water is safe for consumers to drink. Disinfection of water means the destruction of organisms to such low levels that no infection of disease results, when the water is used for domestic purposes including

drinking (Johnson et al., 2009). Chlorine gas is the disinfection method employed at the water treatment works. The components of the water treatment plant that are involved in the disinfection process can be broken down into the following components: Chlorine supply, chlorine injection, disinfection element and the water treatment works manager.

The chlorine supply unit (as shown in Figure 8.1) controls the pressure of chlorine gas and thereby allows the infusion of regulated chlorine gas pressure from the chlorine injection into the water at the disinfection element. The disinfection element of the water chlorination process is the area where the disinfection takes place by infusing the regulated chlorine into the un-disinfected water present in its contact tanks. The residual chlorine monitor located in the contact tank measures the residual chlorine reading and convey this information back to the chlorine supply. This allows the chlorine supply to control the whole water chlorination process thereby maintaining the safety of the water. The information on the status of the whole water chlorinating process is fed to the water treatment works manager who in turn is responsible for the supervision of the entire water treatment works.



Figure 8.1: Part of the chlorination system of water treatment works.

8.3.2 The Participants Groups

In each evaluation phase, the prototype software editor was used by users who were asked to create hazard and early warning analysis models. During the first phase of the evaluation the users formed two groups. The first group consisted of those who had well established knowledge of the hazard and early warning analysis and the second group was comprised of members of the staff of the drinking water treatment works (i.e. they were domain experts in drinking water treatment plants operations). Each group was composed of four experts. The experts in drinking water treatment plant domain had no experience with the hazard and early warning analysis. An expert from the first group (i.e. with established knowledge of the hazard and early warning analysis) assisted the users of the second group and in the same way, an expert from the second group (i.e. with domain experts in drinking water treatment plant) assisted the user of the first group. Thus, three users of each group have used the prototype editor and complete the SUS questionnaires.

For the second phase of the evaluation, the participants were the same four hazard and early warning analysis domain experts participated in first phase of the evaluation.

8.4 1st Evaluation Phase

The main tasks of the evaluation at this phase were to:

- 1) Prepare the two SUS questionnaires with ten questions each. As mentioned in section 8.2.1, one questionnaire was composed of ten questions focused mainly on the satisfaction aspect of the prototype editor, whereas the second contained questions focused on the effectiveness of the three DSMLs.
- 2) Let the user groups of the drinking water and of the hazard and early warning analysis experts perform an analysis for the disinfection process of the drinking water treatment works using the prototype software editor.
- 3) Pass both SUS questionnaires to the participants, analyse their ratings and calculate the overall SUS score for each individual participant.

At the beginning, questionnaire have been prepared according to the SUS guidelines. A copy of each type of questionnaire is provided in Appendix A.

Next, the experts from the two participant groups were gathered in the drinking water treatment works to participate into a series of introductory

sessions in order to be introduced to the hazard and early warning analysis and in to the operations of the disinfection process. During these sessions the hazard and early warning analysis experts explained the method and introduced the concepts of the analysis as well as the icons/symbols and other features of the user interface of the prototype software editor to the drinking water treatment works experts. They also provided notes that demonstrated the use of the prototype software editor. The drinking water treatment works experts, on the other hand, gave a tour of the operations of the plant and explained in detail the operations of the disinfection process. Then all together formed a team and practiced the use of the editor.


Afterwards, one expert from each participant group was assigned to the other group in order to assist the experts during the sessions where they will be using the prototype software editor to create hazard and early warning analysis models for the disinfection process. The three experts in each group were provided with laptop computers and were asked to perform the analysis with the prototype software editor. The three experts of each group of users were free to ask their assistant any question about the hazard and early warning analysis or about the operation of the disinfection process during the span of the DSML models creation sessions. However, they were not allowed to ask for any guidance on how to create the DSML models in to the prototype software editor. There were 13 sessions that lasted three to four hours each, during which the experts in both groups were creating the hazard and early warning analysis models of the disinfection process. All experts were asked to analyse the same hazard "The water chlorination process should always disinfect the water". Furthermore, they were asked to consider their analysis as complete only when they manage to define at least two hierarchical levels of the disinfection process. This practically means to define a "control structure diagram", with at least two elements having the role of controllers, two types of "Inadequate control actions" for each controller, and two "Process models", as well as at least four "Causal factors" associated to the "Hazard" with at least four "Early warning signs".

Finally, the questionnaires were passed to the experts and were asked to provide a rating to each question. What followed was the calculation of the SUS score for each expert together with the analysis of the results.

8.4.1 Indicative Example

The hazard and early warning analysis models created by one of the experts of the first group of participants during this evaluation phase are shown below as indicative example of their creation process.

As previously mentioned the hazard, which all experts analysed is the "The water chlorination process should always disinfect the water". The first task of the expert in this case was to define the concept of "Hazard" in his models. That was possible by interacting with the AML editor of the prototype editor by completing the following steps:

1. The  graphical icon, which represent the concept of a "Hazard" in the AML modeling language, was selected from the palette of the AML editor and placed in the editor view.
2. The expert then entered the proper values of the properties of the "Hazard" concept, such as its name, effects, label and description of the "Hazard" by interacting with the property view of the user interface as Figure 8.2 shows. In this case, for example, the value of the name of the "Hazard" is "The water chlorination process should always disinfect the water"

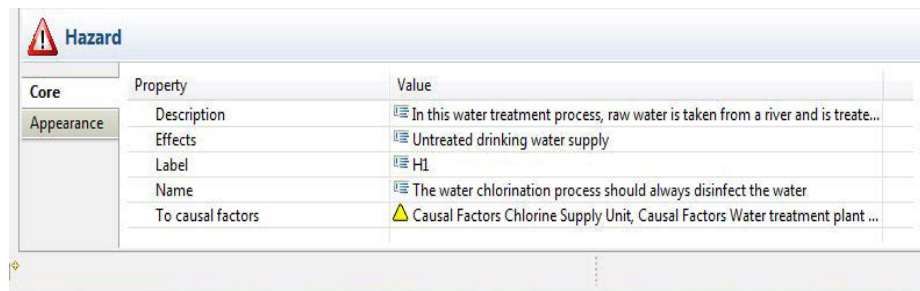





Figure 8.2: The concept of "Hazard" specified using AML editor.

Next, the expert created a model of the "Control structure diagram" of the disinfection process by completing the following steps:

1. He selected the graphical icons ,  and  from the palette of CSML editor which represent the concepts of "Control structure diagram", "Component" and "Link" of the CSML modeling language and created the "Control structure diagram" of the disinfection process. The result was the control structure diagram model that is shown in Figure 8.3.

2. He was able then to provide the information associated to each element of the model by entering the necessary values into the property view. For example, Figure 8.4 depicts the property view of a "Component" element of the control structure diagram model at the time where the expert is ready to assigns the "Controller" value into its role attribute.

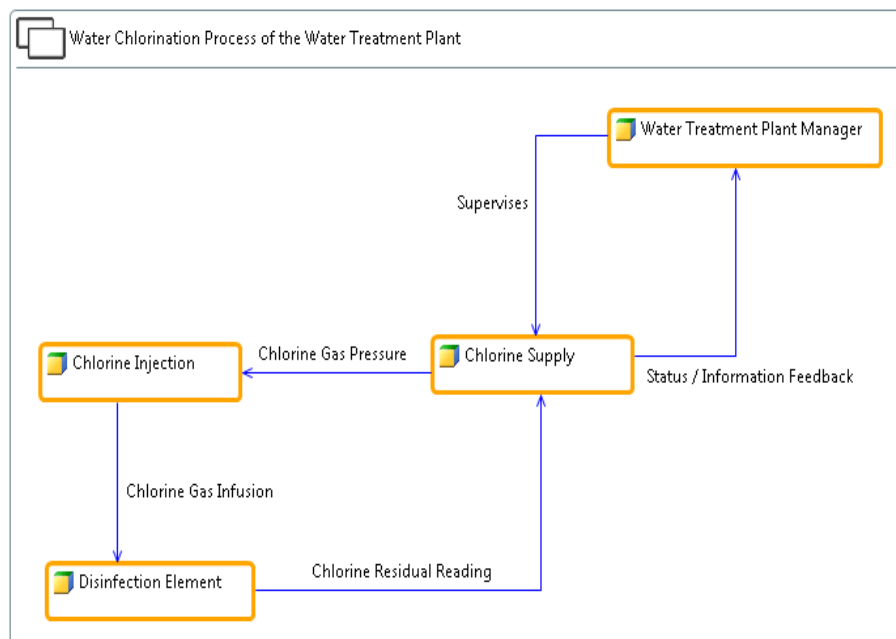


Figure 8.3: Defining a "Control structure diagram" using the CSML editor.

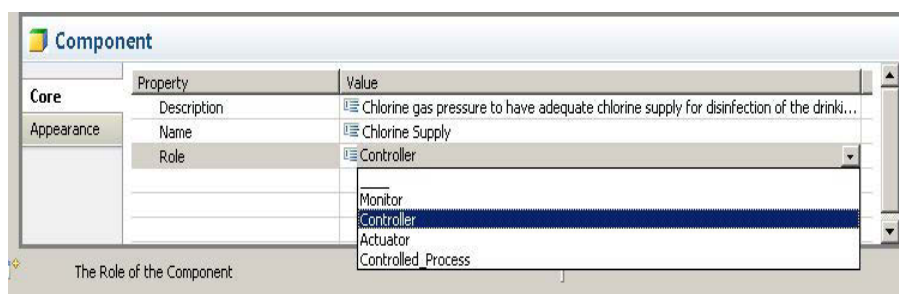








Figure 8.4: The property view of a component in a "Control structure diagram" using the CSML editor.

Next, the expert created a model of a feedback control process by selecting the graphical icons , , ,  and , which represents the concepts of "Controller", "Actuator", "Sensor", "Controlled process" and "Link" of the ICAML modeling language, from the palette of the ICAML editor. The expert then was able to define the "Inadequate control actions" in his models by selecting first the graphical icon , which represents the

concept of "Inadequate control action", from the palette of the ICAML editor and then placing it into the element of the feedback control process model that has the role of the controller. A model of a feedback control process is shown in Figure 8.5. The controller in this model contains an "Inadequate control action" of the type "incorrectly provided".

He was also capable to define the values of the properties, such as their name, description etc., of the elements in his feedback control process model by entering them into the proper field of the property view. For example, Figure 8.6 shows the property view of the "Inadequate control action" elements embedded into the controller of the feedback control process model. In this figure the expert was able to specify the name of an "Inadequate control action" as "Regulated Chlorine Gas Pressure" and to specify its type as "Incorrectly Provided".

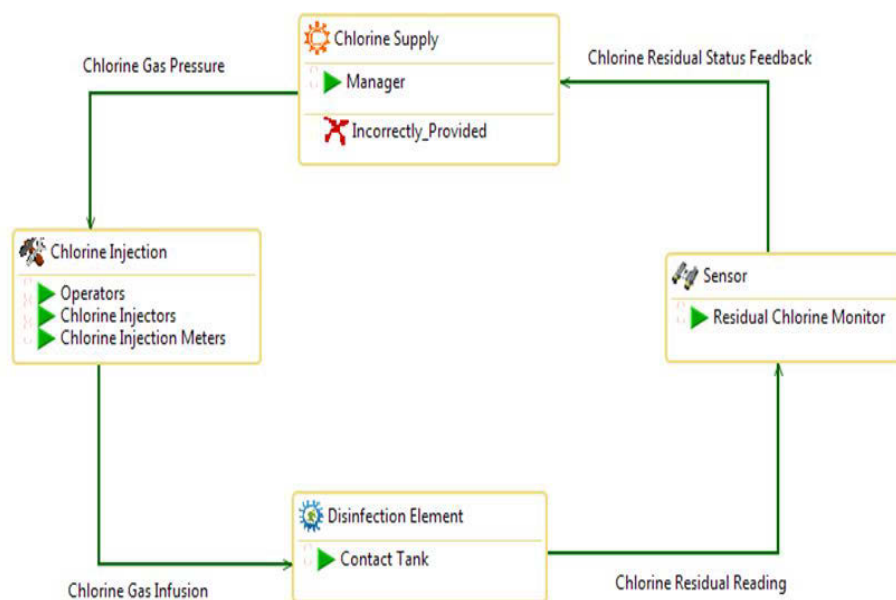


Figure 8.5: Assigning "Inadequate control actions" using the ICAML editor.

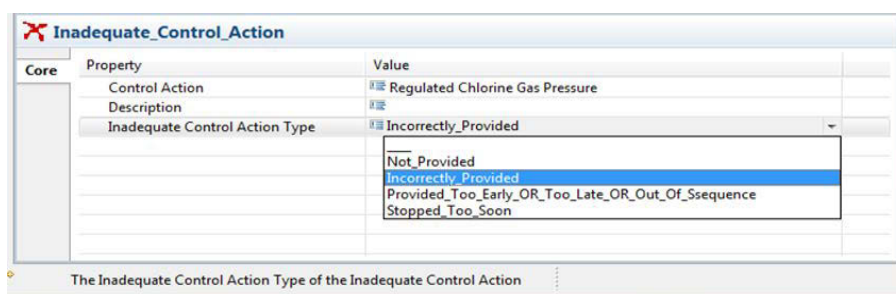


Figure 8.6: The "Inadequate control action" dialogue in ICAML editor.

The expert then defined the "Process models" of each controller of his

feedback control process models. Figure 8.7 and depicts how this was done. Specifically, by interacting with the property view of the controllers in the ICAML editor the expert has been able to define the values for the "process model inputs", "process model outputs", "process model rules" attributes of the controller element. In addition, Figure 8.8 depicts how the domain expert has updated the "Process model" of the "Controller" with the "Early warning sign".

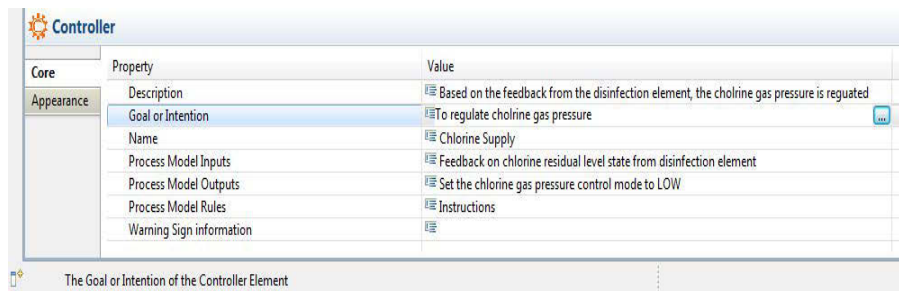


Figure 8.7: Defining the "Process model" used to control water chlorination process.

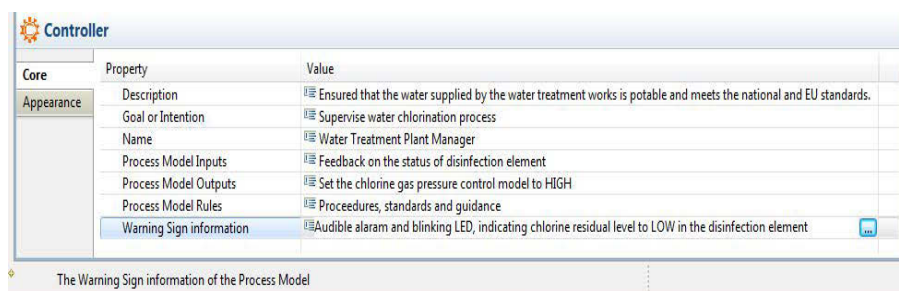





Figure 8.8: Updating the "Process model" used to control water chlorination process.

Next, the expert defined the hazard models for the disinfection process of the water treatment works. In order to do this he selected the graphical icon  from the palette of the AML editor and created as many as need "Causal factor" elements in to the editor view. He then selected the icon  from the pallet to create the links of his hazard model. He used the links to connect the "Causal factors" elements between them and with the "Hazard" element which was represented in to the editor view by the element with the graphical icon . The result was a tree like hazard model as Figure 8.9 shows. He was able then to define the name, description and generic control flaw types properties of the "Causal factor" elements as Figure 8.10 shows .

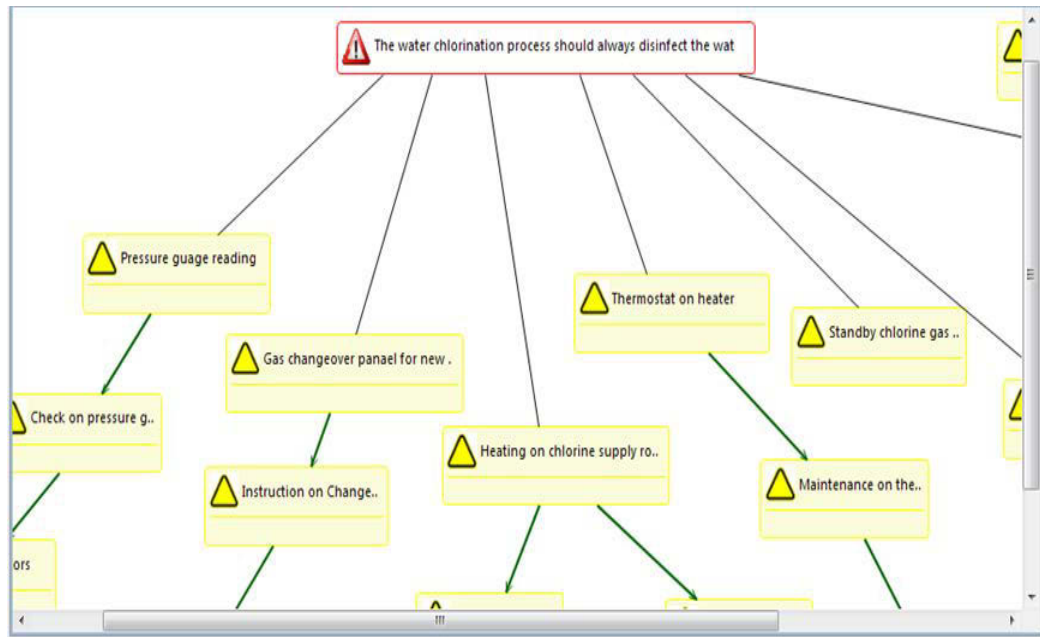


Figure 8.9: Assigning control process flows using the AML editor.

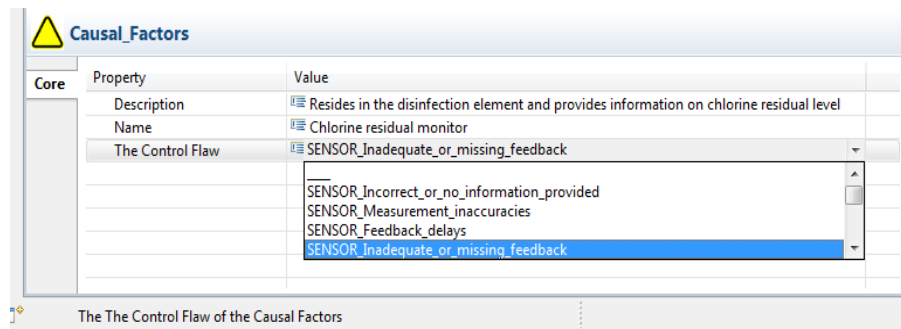


Figure 8.10: Assigning of possible generic control process flow type.

Next, the expert defined the "Early warning signs" and their attributes for each "Causal factor" in the hazard model. In order to do this he has selected the graphical icon from the palette of the AML editor and placed it into "Causal factor" element of the hazard model for the disinfection process of the water treatment works. As result, the hazard model has been updated with the "Early warning signs" of each "Causal factor" of the "Hazard" as shown in Figure 8.11. The expert has interacted with the property view of the "Early warning signs" in the AML editor to enter the values of the "Early warning sign" properties as Figure 8.12 shows.

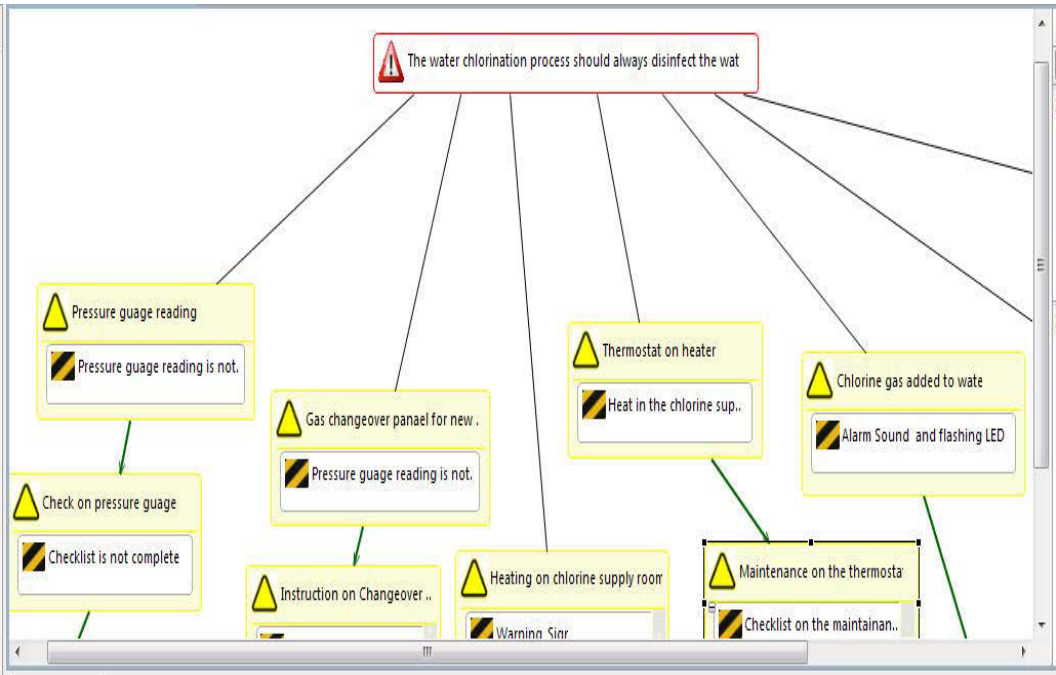



Figure 8.11: Specifying "Early warning signs" using AML editor.

Warning_Sign		
Core	Property	Value
Appearance	Description	Water chlorination process
	Earlywarning Sign	Audible alarm and blinking LED
	Early Warning Sign Justification Information	
	Earlywarningsign Means Of Data Transfer	Alarm sound and flashing light
	Earlywarningsign Receiver	Water treatment works manager
	Earlywarningsign Source	Residual chlorine detector in the disinfection element
	Earlywarningsign Timestamp	20101222113004
	Name	ResidualChlorineLevel

Figure 8.12: Attributes of "Early warning sign".

Next, the expert generated computer code in xml and java format by clicking on the code generation icon  placed on the menu bar of a prototype software editor. The snippet of the computer code generated from the hazard model shown in Figure 8.11 is shown in Figures 8.13 and 8.14.


```

- <Analysis_System>
- <Hazard name="The water disinfection process should always disinfect the water" Label="H1" Effects="Untreated drinking water supply" Description="In this water treatment process, raw water is taken from a river and is treated with chlorine">
- <Causal_Factors name="Pressure guage reading" Description="Inappropriate pressure guage reading on the changeover control panel for a new drum" The_Flaw="SENSOR_Incorrect_or_no_information_provided">
  <Warning_Sign source="Pressure guage reading on the changeover control panel (in Chlorine supply unit of the water chlorination process)" Means_of_data_transfer="Observation" Receiver="Water treatment works Operator (person responsible for checking the pressure guage on the changeover panel)"
  Timestamp="20101223152609" Waring_sign="Pressure guage reading is not between 3.5 and 6 bar ">
  </Warning_Sign>
</Causal_Factors>
- <Causal_Factors name="Check on pressure guage" Description="Inconsistent and incomplete checklist on pressure guage" The_Flaw="CONTROLLER_Process_Model_Inconsistent_incomplete_or_incorrect">
  <Warning_Sign source="Changeover control panel in Chlorine supply unit of the water chlorination process" Means_of_data_transfer="Observation" Receiver="Water treatment works Manager (person responsible for keeping track on the checklist for pressure guage)" Timestamp="20101223155611" Waring_sign="Checklist is not complete"> </Warning_Sign>
</Causal_Factors>
- <Causal_Factors name="Thermostat on heater" Description="Ineffective thermostat on heater" The_Flaw="CONTROLLER_Inappropriate_ineffective_or_missing_control_action">
  <Warning_Sign source="Thermostat on heater" Means_of_data_transfer="Feel" Receiver="Water treatment works Manager" Timestamp="20101223110343" Waring_sign="Heaters thermostat not working">
  </Warning_Sign>
</Causal_Factors>
</Hazard>
</Analysis_System>

```

Figure 8.13: Snippet of XML code generated using AML editor.

```

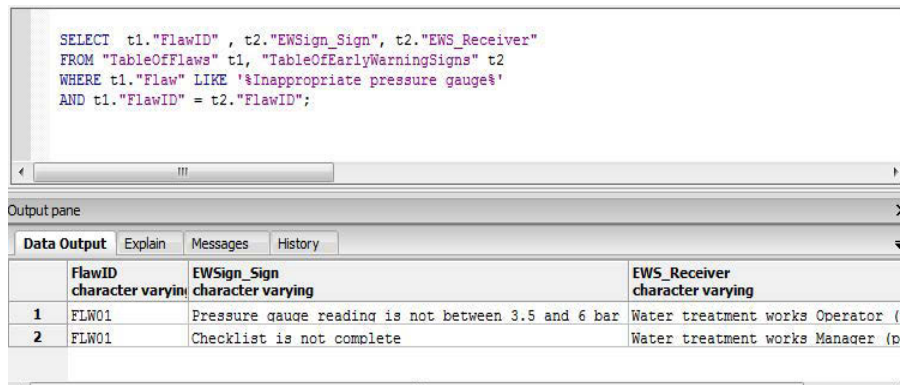
3
4 public class Analysis{
5   class hazard {
6     String name = "The water disinfection process should always disinfect the water";
7     String Lbl = "H1";
8     String Eff = "Untreated drinking water supply";
9     String Desc = "In this water treatment process, raw water is taken from a river a
10  }
11
12  class Causalfactor1 {
13    hazard h = new hazard();
14    String Hazard_name = h.name;
15    String Hazard_label = h.Lbl;
16    String Hazard_Effect = h.Eff;
17    String Hazard_Descripton = h.Desc;
18
19    String CausalFactor_Flaw = "SENSOR_Incorrect_or_no_information_provided";
20    String CausalFactor_Name = "Pressure guage reading";
21    String CausalFactor_Description = "Inappropriate pressure guage reading on the cha
22
23    String EarlyWaring_sign = "Pressure guage reading is not between 3.5 and 6 bar";
24    String EarlyWaringSign_source = "Pressure guage reading on the changeover control
25    String EarlyWaringSign_means_of_data_transfer = "Observation";
26    String EarlyWaringSign_receiver = "Water treatment works Operator (person responsil
27    String EarlyWaringSign_timestamp = "20101223152609";
28  }
29
30  class Causalfactor2 {}
47
48  class Causalfactor3 {}
65 }
66

```

Figure 8.14: Snippet of Java code generated using AML editor.

Next, the expert was able to store the generated XML code into the database

layer of the editor and was also able to query the saved information by writing SQL statements in to the SQL Editor of the Database Explorer Editor. Figure 8.15 shows an query statement written and executed by the expert to find out the warning signs related to the flaw *"Inappropriate pressure gauge reading on the changeover control panel for a new drum"*.



The screenshot shows a SQL query in a text editor window. Below the editor is an 'Output pane' with tabs for 'Data Output', 'Explain', 'Messages', and 'History'. The 'Data Output' tab is selected, displaying a table with the results of the query.

```
SELECT t1."FlawID" , t2."ENSign_Sign", t2."EWS_Receiver"
FROM "TableOfFlaws" t1, "TableOfEarlyWarningSigns" t2
WHERE t1."Flaw" LIKE '%Inappropriate pressure gauge%'
AND t1."FlawID" = t2."FlawID";
```

	FlawID character varying	ENSign_Sign character varying	EWS_Receiver character varying
1	FLW01	Pressure gauge reading is not between 3.5 and 6 bar	Water treatment works Operator (p
2	FLW01	Checklist is not complete	Water treatment works Manager (pe

Figure 8.15: A Query performed using database explorer editor.

8.4.2 SUS Data Collection and Analysis

After the creation of the hazard and early warning analysis models the experts were asked to complete the two SUS questionnaires. Figures 8.16 and 8.17 depict a copy of each SUS questionnaire that have been completed by one of the experts. What followed that was the calculation of the SUS score of each questionnaire.

Tables 8.1 to 8.6 and 8.7 to 8.12 shows the ratings given by each expert to the questions of the questionnaires depicted in Figures 8.16 and 8.17, as well as the SUS score and how it was calculated.

For example, Table 8.1 shows the ratings given by the expert to each question of the questionnaire of Figure 8.16. It also shows the question scores, as well as the total SUS score. Finally, Tables 8.13 and 8.14 show the overall SUS scores of each experts ratings on the satisfaction aspect of prototype editor and of the effectiveness aspect of the DSMLs constructs.

The Usability Evaluation Questionnaire

For each statements below circle the rating that indicates your assessment on the use of prototype software editor for the hazard and early warning sign analysis.

1. I found the prototype software editor to be user friendly.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

2. It was not easy to use the prototype software editor.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

3. I found using of the prototype software editor for early warning sign analysis to be very useful.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

4. Conducting hazard and early warning sign analysis with the prototype software editor does not saves time.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

5. While using the prototype software editor, I found that I could reenter the information stored into the models quickly and easily.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

6. I found the system very cumbersome to use.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

7. I feel that most non hazard experts can learn to use this tool quickly.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

8. I needed to learn a lot of things before I could get going with this system.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

9. I feel that I would like to conduct hazard and early warning sign analysis using the prototype software editor more frequently .

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

10. Overall, I found the prototype software tool is unnecessarily complex.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

Figure 8.16: User satisfaction ratings over the satisfaction aspect of the usability of the prototype editor.

The Usability Evaluation Questionnaire - Effectiveness Aspect

For each statements below circle the rating that indicates your assessment on the use of hazard and early warning sign analysis DSMLs using the prototype software editor.

1. I was able to define a control structure diagram model of the reference system using the constructs of the modeling language.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

2. It was not easy to depict all necessary elements of the reference system into the control structure diagram.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

3. I found that the constructs of the modeling language allowed me to create all elements of the feedback control process model.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

4. I could not figure out how to specify the inadequate control actions into the controller element of the feedback control process model.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

5. I found it quite simple to define and to update the process model of the controller in the feedback control processes model.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

6. I could not find a type of generic control flow that I was looking for in order to define the flow in the feedback control process.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

7. I found that I was able to define the concept of a hazard and its causal factor elements without any difficulty.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

8. While creating the hazard model, I could not find out how to specify the warning signs associated to each causal factor.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

9. Overall, I found that the constructs of the modeling languages enables me to complete all the tasks of the hazard and early warning analysis.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

10. I found the the graphical icons representing the constructs of the modeling language are not appropriate.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

Figure 8.17: User satisfaction ratings over the effectiveness of the DSMLs constructs.

8.4. 1st Evaluation Phase

Table 8.1: Calculation of the SUS score based on the User 01 ratings of the questionnaire shown in Figure 8.16.

Participant	Questions								SUS score = Y+Z
User 1	Odd Numbered Questions	Q1	Q3	Q5	Q7	Q9			
	User Rating	4	4	4	4	5	a = Total Score of Odd Numbere d Questions	Y = a x 2.5	
	Question Score (User Rating Number – 1)	3	3	3	3	4	16	40	
	Even Numbered Questions	Q2	Q4	Q6	Q8	Q10			
	User Rating	2	1	2	1	1	b = Total of Even Numbere d Questions	Z = b X 2.5	
	Question Score (5 - User Rating Number)	3	4	3	4	4	18	45	
									85

Table 8.2: Calculation of the SUS score based on the User 02 ratings of the questionnaire.

Participant	Questions								SUS score = Y+Z
User 2	Odd Numbered Questions	Q1	Q3	Q5	Q7	Q9			
	User Rating	5	4	4	4	4	a = Total Score of Odd Numbered Questions	Y = a x 2.5	
	Question Score (User Rating Number – 1)	4	3	3	3	3	16	40	
	Even Numbered Questions	Q2	Q4	Q6	Q8	Q10			
	User Rating	2	2	2	1	2	b = Total of Even Numbered Questions	Z = b X 2.5	
	Question Score (5 - User Rating Number)	3	3	3	4	3	16	40	
									80

8.4. 1st Evaluation Phase

Table 8.3: Calculation of the SUS score based on the User 03 ratings of the questionnaire.

Participant	Questions								SUS score = Y+Z
User 3	Odd Numbered Questions	Q1	Q3	Q5	Q7	Q9			
	User Rating	4	5	4	4	4	a = Total Score of Odd Numbere d Questions	Y = a x 2.5	
	Question Score (User Rating Number – 1)	3	4	3	3	3	16	40	
	Even Numbered Questions	Q2	Q4	Q6	Q8	Q10			
	User Rating	2	2	1	1	2	b = Total of Even Numbere d Questions	Z = b X 2.5	
	Question Score (5 - User Rating Number)	3	3	4	4	3	17	42.5	
									82.5

Table 8.4: Calculation of the SUS score based on the User 04 ratings of the questionnaire.

Participant	Questions								SUS score = Y+Z
User 4	Odd Numbered Questions	Q1	Q3	Q5	Q7	Q9			
	User Rating	4	4	4	2	4	a = Total Score of Odd Numbered Questions	Y = a x 2.5	
	Question Score (User Rating Number – 1)	3	3	3	1	3	13	32.5	
	Even Numbered Questions	Q2	Q4	Q6	Q8	Q10			
	User Rating	1	2	1	4	2	b = Total of Even Numbered Questions	Z = b X 2.5	
	Question Score (5 - User Rating Number)	4	3	4	1	3	15	37.5	
									70

8.4. 1st Evaluation Phase

Table 8.5: Calculation of the SUS score based on the User 05 ratings of the questionnaire.

Participant	Questions								SUS score = Y+Z
User 5	Odd Numbered Questions	Q1	Q3	Q5	Q7	Q9			
	User Rating	4	4	4	2	4	a = Total Score of Odd Numbered Questions	Y = a x 2.5	
	Question Score (User Rating Number – 1)	3	3	3	1	3	13	32.5	
	Even Numbered Questions	Q2	Q4	Q6	Q8	Q10			
	User Rating	2	2	1	4	2	b = Total of Even Numbered Questions	Z = b X 2.5	
	Question Score (5 - User Rating Number)	3	3	4	1	3	14	35	
									67.5

Table 8.6: Calculation of the SUS score based on the User 06 ratings of the questionnaire.

Participant	Questions								SUS score = Y+Z
User 6	Odd Numbered Questions	Q1	Q3	Q5	Q7	Q9			
	User Rating	4	4	4	2	4	a = Total Score of Odd Numbered Questions	Y = a x 2.5	
	Question Score (User Rating Number – 1)	3	3	3	1	3	13	32.5	
	Even Numbered Questions	Q2	Q4	Q6	Q8	Q10			
	User Rating	1	1	1	4	2	b = Total of Even Numbered Questions	Z = b X 2.5	
	Question Score (5 - User Rating Number)	4	4	4	1	3	16	40	
									72.5

8.4. 1st Evaluation Phase

Table 8.7: Calculation of the SUS score based on the User 01 ratings of the questionnaire shown in Figure 8.17 - Effectiveness Aspect.

Participant	Questions								SUS score = Y+Z
User 1	Odd Numbered Questions	Q1	Q3	Q5	Q7	Q9			
	User Rating	5	5	4	4	4	a = Total Score of Odd Numbered Questions	Y = a x 2.5	
	Question Score (User Rating Number – 1)	4	4	3	3	3	17	42.5	
	Even Numbered Questions	Q2	Q4	Q6	Q8	Q10			
	User Rating	1	2	1	1	2	b = Total of Even Numbered Questions	Z = b X 2.5	
	Question Score (5 - User Rating Number)	4	3	4	4	3	18	45	
									87.5

Table 8.8: Calculation of the SUS score based on the User 02 ratings of the questionnaire - Effectiveness Aspect.

Participant	Questions								SUS score = Y+Z
User 2	Odd Numbered Questions	Q1	Q3	Q5	Q7	Q9			
	User Rating	5	5	4	4	4	a = Total Score of Odd Numbered Questions	Y = a x 2.5	
	Question Score (User Rating Number – 1)	4	4	3	3	3	17	42.5	
	Even Numbered Questions	Q2	Q4	Q6	Q8	Q10			
	User Rating	1	2	1	1	2	b = Total of Even Numbered Questions	Z = b X 2.5	
	Question Score (5 - User Rating Number)	4	3	4	4	3	18	45	
									87.5

8.4. 1st Evaluation Phase

Table 8.9: Calculation of the SUS score based on the User 03 ratings of the questionnaire - Effectiveness Aspect.

Participant	Questions								SUS score = Y+Z
User 3	Odd Numbered Questions	Q1	Q3	Q5	Q7	Q9			
	User Rating	5	5	4	4	4	a = Total Score of Odd Numbered Questions	Y = a x 2.5	
	Question Score (User Rating Number – 1)	4	4	3	3	3	17	42.5	
	Even Numbered Questions	Q2	Q4	Q6	Q8	Q10			
	User Rating	2	2	1	1	2	b = Total of Even Numbered Questions	Z = b X 2.5	
	Question Score (5 - User Rating Number)	3	3	4	4	3	17	42.5	
									85

Table 8.10: Calculation of the SUS score based on the User 04 ratings of the questionnaire - Effectiveness Aspect.

Participant	Questions								SUS score = Y+Z
User 4	Odd Numbered Questions	Q1	Q3	Q5	Q7	Q9			
	User Rating	5	4	3	4	4	a = Total Score of Odd Numbered Questions	Y = a x 2.5	
	Question Score (User Rating Number – 1)	4	3	2	3	3	15	37.5	
	Even Numbered Questions	Q2	Q4	Q6	Q8	Q10			
	User Rating	1	3	1	2	1	b = Total of Even Numbered Questions	Z = b X 2.5	
	Question Score (5 - User Rating Number)	4	2	4	3	4	17	42.5	
							80		

8.4. 1st Evaluation Phase

Table 8.11: Calculation of the SUS score based on the User 05 ratings of the questionnaire - Effectiveness Aspect.

Participant	Questions								SUS score = Y+Z
User 5	Odd Numbered Questions	Q1	Q3	Q5	Q7	Q9			
	User Rating	4	4	2	4	5	a = Total Score of Odd Numbered Questions	Y = a x 2.5	
	Question Score (User Rating Number –1)	3	3	1	3	4	14	35	
	Even Numbered Questions	Q2	Q4	Q6	Q8	Q10			
	User Rating	1	3	1	2	2	b = Total of Even Numbered Questions	Z = b X 2.5	
	Question Score (5 - User Rating Number)	4	2	4	3	3	16	40	
									75

Table 8.12: Calculation of the SUS score based on the User 06 ratings of the questionnaire - Effectiveness Aspect.

Participant	Questions								SUS score = Y+Z
User 6	Odd Numbered Questions	Q1	Q3	Q5	Q7	Q9			
	User Rating	5	4	2	4	4	a = Total Score of Odd Numbere d Questions	Y = a x 2.5	
	Question Score (User Rating Number – 1)	4	3	1	3	3	14	35	
	Even Numbered Questions	Q2	Q4	Q6	Q8	Q10			
	User Rating	1	3	2	2	2	b = Total of Even Numbere d Questions	Z = b X 2.5	
	Question Score (5 - User Rating Number)	4	2	3	3	3	15	37.5	
									72.5

Table 8.13: The overall SUS scores of each expert for the satisfaction aspect of the prototype editor.

Participants	SUS score	
User 01	85.0	Max
User 02	80.0	
User 03	82.5	
User 04	70.0	
User 05	67.5	Min
User 06	72.5	
Mean Value	76.25	

Table 8.14: The overall SUS scores of each expert for the effectiveness of the DSMLs constructs.

Participants	SUS score	
User 01	87.5	Max
User 02	87.5	Max
User 03	85.0	
User 04	80.0	
User 05	75.0	
User 06	72.5	Min
Mean Value	81.25	

8.4.3 Discussion of the 1st Evaluation Phase

As mentioned in section 8.2.1. If the calculated SUS score is in the region of 70s it indicates that the usability is acceptable, whereas if it is in the region of 80s the usability it is considered good and in 90s it is considered as exceptional.

Referring to table 8.13, which depicts the overall SUS scores of each expert for the satisfaction aspect of usability, it can be observed that the score

ranged between 67.5 to 85.0 with mean value the 76.25. This indicates that the experts were in overall satisfied with the features of the prototype editor.

Referring to table 8.14 the ratings, which the experts gave to the questions about the effectiveness of the DSMLs produced SUS scores that ranged between 72.5 to 87.5 having a mean value of 81.25. This, indicates that the constructs of the DSMLs were considered appropriate for completing the tasks of the hazard and early warning analysis.

Analysing the SUS scores derived by Tables 8.1 to 8.6 for the completeness aspect of the usability and 8.7 to 8.12 for the effectiveness aspect of usability, one might observe that the scores of the first three users is slightly higher to the SUS scores of the last three users. This is due to the fact that the first three users belong in to the group of experts with knowledge on the hazard and early warning analysis, whereas the last three users belong in to the group of users with the drinking water treatment works experts.

The deviation in the SUS scores for the first questionnaire was due to the user ratings in questions seven "I feel that most non hazard experts can learn to use this tool quickly" and in eight "I need to learn a lot of things before I could get going with this system" as shown in Tables 8.1 to 8.6. Specifically, the water treatment works experts rated these two questions with numbers, which resulted in lower questions score compared to the hazards and early warning analysis experts. That was expected because these experts needed more effort in order to comprehend the concepts of the domain.

The deviation of the SUS scores in the second questionnaire was due to the user ratings in question seven "I found that I was able to define the concept of hazard and it causal factor elements without any difficulty" as shown in Tables 8.7 to 8.12. The water treatment works experts rated this question with numbers, which resulted in lower questions score compared to the hazards and early warning analysis experts. This deviation can be attributed to the fact that the drinking water treatment works experts had not many chances of practicing the analysis as many times as the hazard and early warning analysis experts had.

8.5 2nd Evaluation Phase

The second phase of the evaluation approach aimed at collecting and analysing the data of the benchmark testes. These data were used to assess the efficiency of the prototype software editor. This phase was concluded in four steps.

In the first step, the experts of the first user group (i.e. the four experts with established knowledge of the hazard and early warning analysis) were asked to participate into the benchmarking tests. For this purpose, a room with the appropriate number of desks was arranged. Notes and photographs of the disinfection process of the water treatment works, which was used again as a reference system for the hazard and early warning analysis models, were available in the room. The experts were allowed to bring their own notes about the reference system taken during their visit in to the drinking water works. Pencils, pens with sheets of paper were provided also. Furthermore, a stop watch was given to each expert to measure the time he consumed for completing each tasks of the analysis. At some point laptops with the prototype editor installed were given to the experts. The hardware configurations for these laptops were almost identical with the processor speed ranging from 1.5GHz to 1.66 GHz, 2 GB RAM memory. All laptops were running the "Windows 7" operating system.

In the second step, the four domain experts were asked to analyse manually the hazard "The water chlorination process should always disinfect the water". They were also asked to record the time which were required to complete the tasks of the analysis, by recording the start and end times using their stop watches. During this step the experts did not had the laptops with the prototype editor in their disposal. The outcome of this step was a set of benchmark times for each expert.

In the third step, the four domain experts were asked to analyse the same hazard with the use of the prototype editor and to record the time required to finish the each task of the analysis. The result of this step was a set of test times for each expert.

For the second and third step, the analysis of the expert was considered as complete based on the definition of the same elements that were mentioned in section 8.4, which include the definition of a "Control structure diagram", with two elements having the role of controllers. At least two types of "Inadequate control actions" and two "Process models" should be defined to each "Controller", as well as at least four "Causal factors" associated to the "Hazard" with at least four "Early warning signs".

Finally, in the analysis step the benchmark and test times of each expert were analysed.

8.5.1 Benchmarking Data and Analysis

The time, which the experts recorded during the second and third step of the benchmarking test, are tabulated in Table 8.15. The table includes, the time of each expert to complete the main tasks of the analysis manually and with the support of the prototype software editor.

Table 8.15: Time of each expert to complete the main tasks of the analysis with and without prototype software editor.

Table 8.5: Time of each expert to complete the main tasks of the analysis with and without prototype software editor				
Main Task of the Domain	Description of the Main Task	Experts	Time Consumed (hours)	
			Manually	Prototype Editor
1	Represent the real world elements that comprise the reference system (i.e. define the boundaries and the elements of the reference system) and define the feedback control process(es) responsible for keeping the reference system in a safe state.	User 01	17	6
		User 02	19	5
		User 03	14	5
		User 04	24	8
2	Define the conditions that may lead to accidents (i.e. define the inadequate control actions) given the control processes identified in the previous task.	User 01	9	5
		User 02	9	5
		User 03	7	4
		User 04	13	7
3	Define the flaws and the early warning signs associated in all feedback control processes	User 01	42	11
		User 02	39	10
		User 03	33	8
		User 04	54	15

As Table 8.16 shows, User 03 was the fastest completing the first main task of the analysis manually with 14 hours whereas User 04 was the slowest with 24 hours. On the other hand, User 02 and 03 were the fastest completing the first main task of the analysis with the use of the prototype editor within five hours. The average time for the manual completion of the first main task

of the analysis was 18.5 hours whereas for the analysis with the use of the prototype editor was six hours. Finally, the average increase of efficiency on completing the first main task with the prototype editor is 208%. Tables 8.17 and 8.18, are depicting similar analyses for the second and third main task of the analysis.

Table 8.16: Data analysis of the times needed to complete the first main task of the analysis.

Experts	Time Consumed (hours)					
	Manually (hours)		Prototype editor (hours)		Difference (hours)	Efficiency (%)
User 01	17		6		11	183
User 02	19		5	Min	14	280
User 03	14	Min	5	Min	9	180
User 04	24	Max	8	Max	16	200
Average	18.5		6		12.5	208

Table 8.17: Data analysis of the times needed to complete the second main task of the analysis.

Experts	Time Consumed (hours)					
	Manually (hours)		Prototype editor (hours)		Difference (hours)	Efficiency (%)
User 01	9		5		4	80
User 02	9		5		4	80
User 03	7	Min	4	Min	3	75
User 04	13	Max	7	Max	6	85
Average	9.5		5.25		4.25	80

Table 8.18: Data analysis of the times needed to complete the third main task of the analysis.

Experts	Time Consumed (hours)					Efficiency (%)
	Manually (hours)		Prototype editor (hours)		Difference (hours)	
User 01	42		11		31	281
User 02	39		10		38	290
User 03	33	Min	8	Min	25	287
User 04	54	Max	15	Max	37	312
Average	42		11		31	281

The users benefited the most by the prototype tool during the first and the third main tasks of the domain. Indeed, the average increase of efficiency for the first main task was 208% whereas for the third was 281%. During the second main task the average efficiency increased by 80% , the least compared to the other two main tasks. A reason for that difference was that during the manual analysis, the users created sketch models to define the "Control structure diagram" of the reference system in a manner that resembled the feedback control processes. That "tweak" enabled them to define the "Inadequate control actions" without the need to redraw detailed models of the feedback control processes of the reference system, saving thus time.

On the other hand, the features and functionalities of the prototype software editor, such saving and retrieving their graphical models, the undo and redo functionalities, but more importantly the effects, which the constraints of the DSMLs had to the creation of the models, such as the display of a list of data with the proper values, which were already inserted by the users during their work with other editor views, (i.e. reduced the insertion of the same data values twice) where the main reasons to increase the efficiency.

Table 8.19 depicts the time each expert needed to conduct all tasks of the hazard and early warning analysis, manually and by using the prototype editor. The last column of the table shows the percent of the time efficiency when the software editor was used, whereas its last row depicts the average time based on the gathered data as well as the average efficiency. Figure 8.18. depicts a graph that shows the average time the users spent to complete the

three main tasks of the domain with the use of prototype software editor and manually.

Table 8.19: The time difference between manual analysis and with the use of the prototype editor.

Participants	Manually (h)		With prototype editor (h)		Difference (h)	Efficiency (%)
User 01	68		22		46	209%
User 02	67		20		47	235%
User 03	54	Min	17	Min	37	217%
User 04	91	Max	30	Max	61	203%
Average	70		22.25		47.75	214%

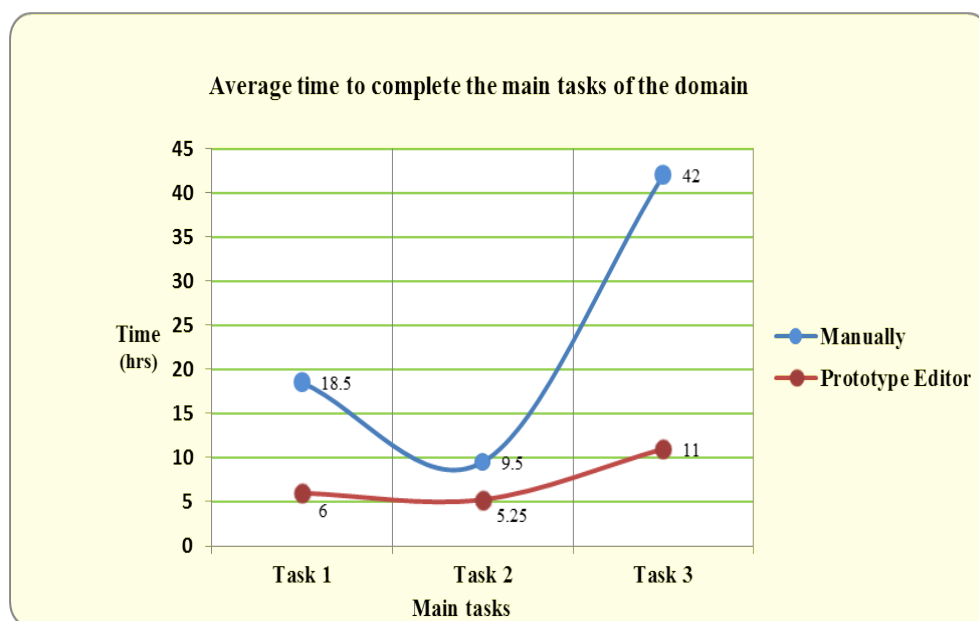


Figure 8.18: Average time to complete the main tasks of the domain.

The minimum time to complete the entire analysis manually was recorded by User 03 whereas the maximum time recorded by User 04. The minimum

time to complete the hazard and early warning analysis via prototype editor was recorded by User 03, whereas the maximum time was recorded by User 04. User 03 recorded the minimum time and the User 04 recorded the maximum time in both steps of the benchmarking. That was not a surprise because User 03 had the most experience of completing the analysis, and the User 04 was the expert who acted as assistant to the group of drinking water works experts during the first evaluation phase of the prototype editor and he didn't created on his own this specific "Hazard" before as the rest of the users did.

8.6 Summary

This chapter described the two phased evaluation approach. During the first phase the objective was to assess the satisfaction of those who used the prototype editor and to assess the effectiveness of the three DSMLs, which were incorporated with the editor. At the second phase, the objective was to assess the efficiency of the prototype software editor with a benchmark test.

SUS questionnaires were used to collect and analyse the data for the first objective and for the second objective benchmark tests that measured the time needed by a group of experts to complete a hazard and early warning analysis have been conducted.

The findings from the data analysis of the SUS questionnaires have shown that the users of the prototype editor have rated its satisfaction with a minimum SUS score the 67.5 and with an average SUS score 76.25, with a maximum possible value the 100, and the effectiveness of the DSMLs with minimum score 72.5 and average score the 81.25. According to the SUS approach, these numbers imply that the satisfaction of the experts for the software editor have been rated, at the worst cases, as 67.5 to 76.25 and that the effectiveness of the DSMLs constructs have been rated from 72.5 to 81.25.

During the second phase, benchmark tests were conducted with four domain experts to measure the time, which they needed to complete an hazard and early warning analysis manually and with the use of the prototype editor. These tests have shown that it took much less time by the experts to complete the analysis with the prototype editor. Specifically, the gain in efficiency was at minimum 203% to an average of 214%. The minimum increase in efficiency was recorded by an expert who, although he was aware of the reference system, he has never conducted a hazard and early warning analysis for it before.

Based on the definition, which states that usability is the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use, and after analysing the results of the evaluation phases on the satisfaction and efficiency of the prototype editor and on the effectiveness of the DSMLs constructs *it is concluded that the overall usability of the proposed DSML based solution has been evaluated as good.*

Learn from yesterday, live for today, hope for tomorrow. The important thing is to not stop questioning.

Albert Einstein

9

Summary and Conclusion

THIS dissertation has introduced three DSMLs and presented a prototype editor in an attempt to enhance the development of the risk model elements of early warning systems. The domain of the three DSMLs is a systems theoretic hazard and early warning analysis approach. Each DSML contains a number of constructs, which enables its users to apply each task of the domain via the creation of graphical models in to the prototype editor. The effectiveness of the DSMLs, as well as the usability of their prototype editor, have been assessed using usability evaluation and benchmarking tests approaches. This chapter will summarise the dissertation and it will discuss the claimed contribution. Thereafter, it will introduce future research directions and expand on some points of self reflection. The chapter ends with final remarks.

9.1 Summary of the Thesis

The thesis began with **Chapter 1** which introduced the main areas of research undertaken. It presented the context of this research, which deals with the creation of computer based risk knowledge elements of early warning systems dealing with critical infrastructures safety issues. The problem statement, research objectives and questions were presented. In particular, selecting of DSM technologies that can be helpful to create a new graphical modeling language, which can be used by an analyst to perform hazard and early warning analysis.

Chapter 2 introduced the concepts and the methods, which were used in the domain of the proposed solution. Specifically, concepts such as accident, hazard and early warning were defined. In addition, a hazard and early warning analysis approach, which integrates the executional steps of STPA hazard

analysis and EWaSAP early warning analysis was presented. The chapter also described all those concepts of the hazard and early warning analysis approach, which are used for the creation of a DSM solution.

In **Chapter 3** the fundamental concepts of DSM were described and the difference between DSML and DSPL was clarified. In this chapter, the possible alternative strategies of developing a DSML solution were presented. These include, the extension of an existing STPA DSML, the refinement of the SysML meta-model (using the UML profiling mechanism) and the development from scratch. A description of the ideas of concrete syntax and abstract syntax was also provided together with meta-modeling and model transformation tools and approaches.

Chapter 4 surveyed the existing DSMLs, which are using concepts related with the domain of the problem of this dissertation. In particular, SOPHIA , EAST-ADL, Stream-Oriented DSML, and OpenCOM and Transition Diagrams DSML were reviewed. From the survey it became obvious that there was not any STPA DSML in existence. Thus, the alternative of extending an existing DSML was eliminated. From the survey, it was also found that there was a lack of methodological guidelines and tool support as well as of code generation mechanisms for the development of DSML with the refinement approach. Based on the findings of the literature review, it was decided to develop a hazard and early warning analysis DSML from scratch.

Chapter 5 covered the software technologies such as EVL, EGL, and modeling frameworks such as EMF, GEF that were used to design and develop the hazard and early warning analysis DSML.

Chapter 6 presented the specifications of 3 DSMLs which combined enables create the proposed solution to the problem. These DSMLs are the CSML, ICAML and AML. Their combined purpose is to allow its user to complete the 3 main tasks or 7 subtasks of the domain. The abstract syntax and concrete syntax of each DSML was presented in this chapter together with their constraints. Finally, the chapter presented an architecture of the hazard and early warning analysis DSML.

Chapter 7 presented the design and architecture of the hazard and early warning analysis editor. Its architecture consists of four layers. The first layer

contains the CSML editor, ICAML editor, and AML editor and a database explorer editor. The second layer contains the abstract syntax of the hazard and early warning analysis DSMLs and their user model validators. The third layer contains three template based code generators, and finally the fourth layer contains the database management system where the users models and the generated code are saved. In addition, the chapter presented the workflow on how to conduct hazard and early warning analysis using prototype software editor.

Finally, **Chapter 8** presented the evaluation of the hazard and early warning analysis DSMLs. The evaluation consisted of two phases. In the first phase, the satisfaction aspect of the usability of the hazard and early warning analysis editor and of the effectiveness aspect of the DSMLs were assessed using the SUS usability evaluation method. In the second phase, the effectiveness aspect of the usability of the prototype software editor was measured with a benchmark test. The chapter then presented the data obtained by each phase of the evaluation and the results of their analysis. It was concluded that the overall usability on the use of hazard and early warning DSMLs was good.

9.2 Contribution of this Research

The novel scientific contributions of this research are the CSML, ICAML and AML modeling languages. The CSML language is comprised of the "Control structure diagram" and "Component" elements and a "Link" relation. The purpose of the CSML is to facilitate the analysts in creating the control structure diagram models of their reference systems and to assign the role of each element within the model.

The ICAML language consisted by six elements, namely the "Controller", "Actuator", "Controlled process", "Process Model", "Sensor", "Inadequate control action" and four relations namely the "Sensor_TO_Controller_link", "Controller_TO_Actuator_Link", "Actuator_TO_ContProc_Link", and "ContProc_TO_Sensor_Link" relations. Its purpose is twofold; the first is to facilitate the analysts in creating models of the feedback control processes within the reference system and to define the "Inadequate control actions" elements of each element who had the role of controller in the models; the second is to enable the users of the DSMLs to define and update the "Process Models"

of each controller.

The AML language consists of the "Hazard", "Causal factor" and "Early warning sign" elements and by the "Hazard_To_CausalFactors_Link" and "Link" relations. Its purpose is to enable analysts to create the causal models of possible accidents of the reference system and to define their associated early warning signs.

By combining these three modeling languages together under one prototype editor, thanks to a set of DSM technologies, the three graphical DSMLs enabled a set of analysts, who participated in to a usability and benchmarking tests, to conduct their hazard and early warning analysis models in a usable manner. The data, which were collected by the usability and benchmark tests were presented in this dissertation.

In particular, during the usability tests the average SUS score for the effectiveness of the three DSMLs was 81.25, which indicates that the modeling languages were considered as good. In addition, the average SUS score about the satisfaction of the users who used the prototype editor was 76.25 which indicates that the DSM technologies that were combined have developed a satisfactory tool. Finally, the results of the benchmark test has shown that the average increase of efficiency which the prototype editor provided was 214%. In overall, these results support the hypothesis of this thesis.

9.3 Critical Remarks

One critical remark of the proposed DSMLs and of their prototype editor is that of the previous theoretical knowledge. Specifically, the users should have a very good understanding of the STAMP accident model as well as of the STPA and EWaSAP approaches. A user who does not have this prior knowledge, will not comprehend the meaning of the constructs of the modeling languages.

Another remark is that the attributes of the elements of the DSMLs and their values are predefined. This means that the users neither can customise or create new properties other than what was defined in the meta-models of the languages, nor can they change the predefined set of values from the attributes of the elements. On the other hand, this can be addressed by those who have the skills to modify the meta-models of the modeling languages.

In reality, a system which is described by one control structure diagram and a set of feedback control processes may have more than one hazards. The limitation of the prototype editor, in its current version, is that it enables

its users to define just a single and not multiple hazard models using the AML modeling language. This, however, can be addressed by a major customisation of the GMF framework.

9.3.1 On the Potential Usefulness of the Research

As described in sections 7.2.3 and 8.4.1 the models created by using the constructs of the three DSMLs can generate computer code. Indeed, the code generation feature was in fact a major factor for selecting a DSM based solution to the problem of defining the risk knowledge element of early warning systems. Specifically, Figures 8.13 and 8.14 have shown the snippet of XML and JAVA code generated by the hazard analysis model which was created by the constructs of the DSMLs that this dissertation has introduced. Thus, it has been shown that the three DSMLs have successfully raised the level of abstraction of the problem domain from the code level to the graphical model level. Therefore, the three DSMLs that were introduced in this dissertation and their prototype software editor may provide useful services in the creation process of the risk knowledge element of computer based early warning systems.

Another useful aspect of the results of this research is that the prototype software editor of the DSMLs is the first that supports the graphical creation of hazard analysis models based on STPA systems theoretic hazard analysis technique. As mentioned in section 1.2. One contributing factor for not using contemporary techniques is the fact that these have only been in existence for a few years and, hence, have not been fully comprehended and adopted by the majority of analysts. Another contributing factor may be the absence of software editors for the contemporary hazard analysis. Indeed, there are no software editors for the STPA hazard analysis nor for the hazard and early warning analysis (i.e. the combination of STPA with EWaSAP). On the other hand it was shown, with the results of the benchmark tests, that the efficiency of the experts who used the prototype editor to create hazard and early warning analysis models has increased by 214% per average. Thus, the prototype editor presented herein and its future research can contribute in making STPA and EWaSAP approaches more popular among professionals in this domain.

9.4 Future Research Directions

Three DSMLs were introduced in this dissertation, which combined allow analysts to conduct the steps of the hazard and early warning analysis in a

usable and effective manner. This proposed DSM solution has paved the way to the following future research directions.

9.4.1 Web Based Accessibility of the DSMLs

The hazard and early warning analysis DSML presented in this research are meant to be used by domain experts after installing and configuring the prototype software editor on their local computer machine. However, the possibility of accessing their work from the internet will enable its users to work from different locations with different computers.

One way of achieving this is by changing the design of the prototype hazard and early warning analysis editor by integrating the Rich Ajax platform (RAP) (Lange, 2008). RAP is an open source framework developed by the Eclipse Foundation, which can be used to develop web applications based on existing Eclipse technologies. RAP utilise the RAP Widget Toolkit (RWT) libraries that replaces the SWT, which is used as a base for the Draw2D framework that is a part of GEF. The same Java application code that was originally developed for SWT can also run with RWT, but instead of creating a display on the local computer screen, the RAP application is acting as a web server. Web clients that connect to this web server will see a display in their web browser that mimics the original SWT display. The RAP application is typically installed in a web container like Tomcat. The RWT library creates HTML and Java Script code for the web client to send updates to the web client (Kasemir et al., 2011). In addition, a setup of multi-tenant database over traditional single tenant database would share the same resources for each user of the DSMLs.

9.4.2 Creating Hazard and Early Warning Analysis Models in a Collaborative Manner

Another important future research direction is to design and develop the necessary technologies, which will enable many analysts who are scattered in different geographical locations to work simultaneously on the same DSML model over the internet. With the integration of such technologies the hazard and early warning analysis editor could be used as a collaborative tool. Currently such DSM technologies do not exist. One way of achieving this is by adapting Operational Transformation (OT) (Ellis and Sun, 1998) technologies along with other web technologies. The OT techniques can allows concurrent operations to be executed in any orders by distributed users, and

their final effects are identical. The use of OT techniques does not provide all the required features, however, it can be used as basis for the creation of such a prototype software editor which can be used by different users from their local machine to create models using hazard and early warning DSMLs.

9.4.3 Compare the Updating Process of the Risk Knowledge Element of a Real Early Warning System

This dissertation has shown that the DSMLs enable analysts to define hazard and early warning analysis manner in a usable manner. A future research direction is to measure whether the performance of updating the risk knowledge element of a real early warning system will be increased with the use of the proposed DSMLs. One way of doing this is by setting a benchmark of the updating risk knowledge element process of a real early warning system without using the DSMLs and then testing the updating process, this time with the DSMLs, against these benchmarks.

9.4.4 Reasoning Support

Another future research direction is to provide a reasoning support in to the prototype software editor in order to enable its users to assess the likelihood of a hazard given specific data over their created models. This can be addressed by integrating a Bayesian Network (BN) (Jensen, 1996), Hidden Markov Models (Rabiner and Juang, 1986) or Artificial Neural Network (Hsu et al., 1995) modules in to the prototype software editor. For example, in the case of integrating a BN reasoning engine the user will be able to insert conditional probability values in each causal factor in to his AML models in order to calculate the likelihood of the occurrence of the hazard in his models, given the presence of a set early warning signs, which may be interpreted by the BN as evidence. It would be interesting to find out which reasoning module, or a combination of these, will help analysts to better understand the omissions of their models by studying the outcome of the reasoned modules.

9.5 Concluding Remarks

The thesis addressed two key research questions. The first question was about the way of integrating DSML technologies in order to create an effective hazard and early warning analysis tool with code generation support. The second question was about the number of the DSML languages and their constructs, which are required so that to enable analysts to complete the tasks of the hazard and early warning analysis.

The first question was addressed in Chapter 5, which presented a set of open source software technologies that were integrated for the creation of prototype hazard and early warning analysis editor. The second question was addressed in Chapter 6, which presented an architecture of three graphical modeling languages which consists of ten elements and seven relations.

The author envisages that future advancements in DSM technologies will further improve the model creation, code generation and integration of the hazard and early warning analysis DSMLs, which were presented in this dissertation. The author hopes that the proposed DSMLs with their prototype editor will influence safety analysts in their work by making STPA and EWaSAP techniques more easy to work with. Furthermore, he hopes that this dissertation will motivate new researchers to develop novel and efficient ways in creating the risk knowledge elements of early warning systems.

AML Analysis Modeling Language.

BN Bayesian Network.

CSML Control Structure Modeling Language.

CSS Cascading Style Sheets.

DSL Domain Specific Language.

DSM Domain Specific Modeling.

DSML Domain Specific Modeling Language.

DSPL Domain Specific Programming Language.

EGL Epsilon Generation Language.

EMF Eclipse Modeling Framework.

EMOF Essential Meta-object Facility.

ETL Epsilon Transformation Language.

EVL Epsilon Validation Language.

EWaSAP Early Warning Sign Analysis Using STPA.

GEF Graphical Editing Framework.

GME Generic Modeling Environment.

GMF Graphical Modeling Framework.

GPL general-purpose language.

HTML Hyper Text Markup Language.

ICAML Inadequate Control Action Modeling Language.

IDE Integrated Development Environment.

IDEF0 Integration Definition for Function Modeling.

ISO International Organization for Standardization.

JDBC Java DataBase Connectivity.

MDA Model Driven Architecture.

MDD Model Driven Development.

MOF MetaObject Facility.

MS Microsoft.

OMG Object Management Group.

QUIS Questionnaire for User Interface Satisfaction.

RAP Rich Ajax platform.

RWT RAP Widget Toolkit.

SQL Structured Query Language.

STAMP Systems Theoretic Accident Models and Processes.

STPA STAMP-Based Hazard Analysis.

SUMI Software Usability Measurement Inventory.

SUS System Usability Scale.

SWT Standard Widget Toolkit.

SysML Systems Modeling Language.

UML Unified Modeling Language.

XML eXtensible Markup Language.

XPath XML Path Language.



Questionnaires

THIS appendix presents the answers to the two questionnaire that was presented to all six experts who participated in evaluation on the satisfaction aspect of the prototype editor on the effectiveness of the three hazard and early warning analysis DSMLs as mentioned in section 8.4.2.

The Usability Evaluation Questionnaire

For each statements below circle the rating that indicates your assessment on the use of prototype software editor for the hazard and early warning sign analysis.

1. I found the prototype software editor to be user friendly.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

2. It was not easy to use the prototype software editor.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

3. I found using of the prototype software editor for early warning sign analysis to be very useful.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

4. Conducting hazard and early warning sign analysis with the prototype software editor does not saves time.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

5. While using the prototype software editor, I found that I could reenter the information stored into the models quickly and easily.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

Figure A.1: User 01 ratings of the SUS questionnaire on satisfaction aspect of prototype software editor.

6. I found the system very cumbersome to use.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2 ✓	3	4	5

7. I feel that most non hazard experts can learn to use this tool quickly.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4 ✓	5

8. I needed to learn a lot of things before I could get going with this system.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1 ✓	2	3	4	5

9. I feel that I would like to conduct hazard and early warning sign analysis using the prototype software editor more frequently .

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5 ✓

10. Overall, I found the prototype software tool is unnecessarily complex.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1 ✓	2	3	4	5

Figure A.2: User 01 ratings of the SUS questionnaire satisfaction aspect of prototype software editor (Page 2).

The Usability Evaluation Questionnaire - Effectiveness Aspect

For each statements below circle the rating that indicates your assessment on the use of hazard and early warning sign analysis DSMLs using the prototype software editor.

1. I was able to define a control structure diagram model of the reference system using the constructs of the modeling language.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

2. It was not easy to depict all necessary elements of the reference system into the control structure diagram.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

3. I found that the constructs of the modeling language allowed me to create all elements of the feedback control process model.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

4. I could not figure out how to specify the inadequate control actions into the controller element of the feedback control process model.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

5. I found it quite simple to define and to update the process model of the controller in the feedback control processes model.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

Figure A.3: User 01 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs.

6. I could not find a type of generic control flaw that I was looking for in order to define the flaw in the feedback control process.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1 ✓	2	3	4	5
-----	---	---	---	---

7. I found that I was able to define the concept of a hazard and its causal factor elements without any difficulty.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	4 ✓	5
---	---	---	-----	---

8. While creating the hazard model, I could not find out how to specify the warning signs associated to each causal factor.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1 ✓	2	3	4	5
-----	---	---	---	---

9. Overall, I found that the constructs of the modeling languages enables me to complete all the tasks of the hazard and early warning analysis.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	4 ✓	5
---	---	---	-----	---

10. I found the the graphical icons representing the constructs of the modeling language are not appropriate.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2 ✓	3	4	5
---	-----	---	---	---

Figure A.4: User 01 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs (Page 2).

The Usability Evaluation Questionnaire - Effectiveness Aspect

For each statements below circle the rating that indicates your assessment on the use of hazard and early warning sign analysis DSMLs using the prototype software editor.

1. I was able to define a control structure diagram model of the reference system using the constructs of the modeling language.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5 ✓

2. It was not easy to depict all necessary elements of the reference system into the control structure diagram.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2 ✓	3	4	5

3. I found that the constructs of the modeling language allowed me to create all elements of the feedback control process model.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5 ✓

4. I could not figure out how to specify the inadequate control actions into the controller element of the feedback control process model.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2 ✓	3	4	5

5. I found it quite simple to define and to update the process model of the controller in the feedback control processes model.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4 ✓	5

Figure A.5: User 02 ratings of the SUS questionnaire on satisfaction aspect of prototype software editor.

6. I could not find a type of generic control flaw that I was looking for in order to define the flaw in the feedback control process.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

<input checked="" type="radio"/> 1	2	3	4	5
------------------------------------	---	---	---	---

7. I found that I was able to define the concept of a hazard and its causal factor elements without any difficulty.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	<input checked="" type="radio"/> 4	5
---	---	---	------------------------------------	---

8. While creating the hazard model, I could not find out how to specify the warning signs associated to each causal factor.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

<input checked="" type="radio"/> 1	2	3	4	5
------------------------------------	---	---	---	---

9. Overall, I found that the constructs of the modeling languages enables me to complete all the tasks of the hazard and early warning analysis.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	<input checked="" type="radio"/> 4	5
---	---	---	------------------------------------	---

10. I found the the graphical icons representing the constructs of the modeling language are not appropriate.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	<input checked="" type="radio"/> 2	3	4	5
---	------------------------------------	---	---	---

Figure A.6: User 02 ratings of the SUS questionnaire on satisfaction aspect of prototype software editor (Page 2).

The Usability Evaluation Questionnaire

For each statements below circle the rating that indicates your assessment on the use of prototype software editor for the hazard and early warning sign analysis.

1. I found the prototype software editor to be user friendly.

**STRONGLY
DISAGREE** **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY
AGREE**

1	2	3	4	5
---	---	---	---	---

2. It was not easy to use the prototype software editor.

**STRONGLY
DISAGREE** **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY
AGREE**

1	2	3	4	5
---	---	---	---	---

3. I found using of the prototype software editor for early warning sign analysis to be very useful.

**STRONGLY
DISAGREE** **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY
AGREE**

1	2	3	4	5
---	---	---	---	---

4. Conducting hazard and early warning sign analysis with the prototype software editor does not saves time.

**STRONGLY
DISAGREE** **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY
AGREE**

1	2	3	4	5
---	---	---	---	---

5. While using the prototype software editor, I found that I could reenter the information stored into the models quickly and easily.

**STRONGLY
DISAGREE** **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY
AGREE**

1	2	3	4	5
---	---	---	---	---

Figure A.7: User 02 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs.

6. I found the system very cumbersome to use.

STRONGLY DISAGREE DISAGREE NEUTRAL AGREE STRONGLY AGREE

1	2	3	4	5
---	---	---	---	---

7. I feel that most non hazard experts can learn to use this tool quickly.

STRONGLY DISAGREE DISAGREE NEUTRAL AGREE STRONGLY AGREE

1	2	3	4	5
---	---	---	---	---

8. I needed to learn a lot of things before I could get going with this system.

STRONGLY DISAGREE DISAGREE NEUTRAL AGREE STRONGLY AGREE

1	2	3	4	5
---	---	---	---	---

9. I feel that I would like to conduct hazard and early warning sign analysis using the prototype software editor more frequently .

STRONGLY DISAGREE DISAGREE NEUTRAL AGREE STRONGLY AGREE

1	2	3	4	5
---	---	---	---	---

10. Overall, I found the prototype software tool is unnecessarily complex.

STRONGLY DISAGREE DISAGREE NEUTRAL AGREE STRONGLY AGREE

1	2	3	4	5
---	---	---	---	---

Figure A.8: User 02 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs (Page 2).

The Usability Evaluation Questionnaire

For each statements below circle the rating that indicates your assessment on the use of prototype software editor for the hazard and early warning sign analysis.

1. I found the prototype software editor to be user friendly.

**STRONGLY
DISAGREE** **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY
AGREE**

1	2	3	4	5
---	---	---	---	---

2. It was not easy to use the prototype software editor.

**STRONGLY
DISAGREE** **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY
AGREE**

1	2	3	4	5
---	---	---	---	---

3. I found using of the prototype software editor for early warning sign analysis to be very useful.

**STRONGLY
DISAGREE** **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY
AGREE**

1	2	3	4	5
---	---	---	---	---

4. Conducting hazard and early warning sign analysis with the prototype software editor does not saves time.

**STRONGLY
DISAGREE** **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY
AGREE**

1	2	3	4	5
---	---	---	---	---

5. While using the prototype software editor, I found that I could reenter the information stored into the models quickly and easily.

**STRONGLY
DISAGREE** **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY
AGREE**

1	2	3	4	5
---	---	---	---	---

Figure A.9: User 03 ratings of the SUS questionnaire on satisfaction aspect of prototype software editor.

6. I found the system very cumbersome to use.

STRONGLY DISAGREE DISAGREE NEUTRAL AGREE STRONGLY AGREE

✗	2	3	4	5
---	---	---	---	---

7. I feel that most non hazard experts can learn to use this tool quickly.

STRONGLY DISAGREE DISAGREE NEUTRAL AGREE STRONGLY AGREE

1	2	3	✗	5
---	---	---	---	---

8. I needed to learn a lot of things before I could get going with this system.

STRONGLY DISAGREE DISAGREE NEUTRAL AGREE STRONGLY AGREE

✗	2	3	4	5
---	---	---	---	---

9. I feel that I would like to conduct hazard and early warning sign analysis using the prototype software editor more frequently .

STRONGLY DISAGREE DISAGREE NEUTRAL AGREE STRONGLY AGREE

1	2	3	✗	5
---	---	---	---	---

10. Overall, I found the prototype software tool is unnecessarily complex.

STRONGLY DISAGREE DISAGREE NEUTRAL AGREE STRONGLY AGREE

1	✗	3	4	5
---	---	---	---	---

Figure A.10: User 03 ratings of the SUS questionnaire on satisfaction aspect of prototype software editor (Page 2).

The Usability Evaluation Questionnaire - Effectiveness Aspect

For each statements below circle the rating that indicates your assessment on the use of hazard and early warning sign analysis DSMLs using the prototype software editor.

1. I was able to define a control structure diagram model of the reference system using the constructs of the modeling language.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

2. It was not easy to depict all necessary elements of the reference system into the control structure diagram.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

3. I found that the constructs of the modeling language allowed me to create all elements of the feedback control process model.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

4. I could not figure out how to specify the inadequate control actions into the controller element of the feedback control process model.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

5. I found it quite simple to define and to update the process model of the controller in the feedback control processes model.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

Figure A.11: User 03 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs.

6. I could not find a type of generic control flaw that I was looking for in order to define the flaw in the feedback control process.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

<input checked="" type="radio"/>	2	3	4	5
----------------------------------	---	---	---	---

7. I found that I was able to define the concept of a hazard and its causal factor elements without any difficulty.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	<input checked="" type="radio"/>	5
---	---	---	----------------------------------	---

8. While creating the hazard model, I could not find out how to specify the warning signs associated to each causal factor.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

<input checked="" type="radio"/>	2	3	4	5
----------------------------------	---	---	---	---

9. Overall, I found that the constructs of the modeling languages enables me to complete all the tasks of the hazard and early warning analysis.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	<input checked="" type="radio"/>	5
---	---	---	----------------------------------	---

10. I found the the graphical icons representing the constructs of the modeling language are not appropriate.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	<input checked="" type="radio"/>	3	4	5
---	----------------------------------	---	---	---

Figure A.12: User 03 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs (Page 2).

The Usability Evaluation Questionnaire

For each statements below circle the rating that indicates your assessment on the use of prototype software editor for the hazard and early warning sign analysis.

1. I found the prototype software editor to be user friendly.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	4	5
---	---	---	---	---

2. It was not easy to use the prototype software editor.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	4	5
---	---	---	---	---

3. I found using of the prototype software editor for early warning sign analysis to be very useful.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	4	5
---	---	---	---	---

4. Conducting hazard and early warning sign analysis with the prototype software editor does not saves time.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	4	5
---	---	---	---	---

5. While using the prototype software editor, I found that I could reenter the information stored into the models quickly and easily.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	4	5
---	---	---	---	---

Figure A.13: User 04 ratings of the SUS questionnaire on satisfaction aspect of prototype software editor.

6. I found the system very cumbersome to use.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

7. I feel that most non hazard experts can learn to use this tool quickly.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

8. I needed to learn a lot of things before I could get going with this system.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

9. I feel that I would like to conduct hazard and early warning sign analysis using the prototype software editor more frequently .

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

10. Overall, I found the prototype software tool is unnecessarily complex.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

Figure A.14: User 04 ratings of the SUS questionnaire on satisfaction aspect of prototype software editor (Page 2).

The Usability Evaluation Questionnaire - Effectiveness Aspect

For each statements below circle the rating that indicates your assessment on the use of hazard and early warning sign analysis DSMLs using the prototype software editor.

1. I was able to define a control structure diagram model of the reference system using the constructs of the modeling language.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

2. It was not easy to depict all necessary elements of the reference system into the control structure diagram.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

3. I found that the constructs of the modeling language allowed me to create all elements of the feedback control process model.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

4. I could not figure out how to specify the inadequate control actions into the controller element of the feedback control process model.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

5. I found it quite simple to define and to update the process model of the controller in the feedback control processes model.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

Figure A.15: User 04 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs.

6. I could not find a type of generic control flaw that I was looking for in order to define the flaw in the feedback control process.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	4	5
--------------	---	---	---	---

7. I found that I was able to define the concept of a hazard and its causal factor elements without any difficulty.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	4	5
---	---	---	--------------	---

8. While creating the hazard model, I could not find out how to specify the warning signs associated to each causal factor.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	4	5
---	--------------	---	---	---

9. Overall, I found that the constructs of the modeling languages enables me to complete all the tasks of the hazard and early warning analysis.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	4	5
---	---	---	--------------	---

10. I found the the graphical icons representing the constructs of the modeling language are not appropriate.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	4	5
--------------	---	---	---	---

Figure A.16: User 04 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs (Page 2).

The Usability Evaluation Questionnaire

For each statements below circle the rating that indicates your assessment on the use of prototype software editor for the hazard and early warning sign analysis.

1. I found the prototype software editor to be user friendly.

**STRONGLY
DISAGREE** **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY
AGREE**

1	2	3	4 ✓	5
---	---	---	-----	---

2. It was not easy to use the prototype software editor.

**STRONGLY
DISAGREE** **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY
AGREE**

1	2 ✓	3	4	5
---	-----	---	---	---

3. I found using of the prototype software editor for early warning sign analysis to be very useful.

**STRONGLY
DISAGREE** **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY
AGREE**

1	2	3	4 ✓	5
---	---	---	-----	---

4. Conducting hazard and early warning sign analysis with the prototype software editor does not saves time.

**STRONGLY
DISAGREE** **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY
AGREE**

1	2 ✓	3	4	5
---	-----	---	---	---

5. While using the prototype software editor, I found that I could reenter the information stored into the models quickly and easily.

**STRONGLY
DISAGREE** **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY
AGREE**

1	2	3	4 ✓	5
---	---	---	-----	---

Figure A.17: User 05 ratings of the SUS questionnaire on satisfaction aspect of prototype software editor.

6. I found the system very cumbersome to use.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1 ✓	2	3	4	5

7. I feel that most non hazard experts can learn to use this tool quickly.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2 ✓	3	4	5

8. I needed to learn a lot of things before I could get going with this system.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4 ✓	5

9. I feel that I would like to conduct hazard and early warning sign analysis using the prototype software editor more frequently .

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4 ✓	5

10. Overall, I found the prototype software tool is unnecessarily complex.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2 ✓	3	4	5

Figure A.18: User 05 ratings of the SUS questionnaire on satisfaction aspect of prototype software editor (Page 2).

The Usability Evaluation Questionnaire - Effectiveness Aspect

For each statements below circle the rating that indicates your assessment on the use of hazard and early warning sign analysis DSMLs using the prototype software editor.

1. I was able to define a control structure diagram model of the reference system using the constructs of the modeling language.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4 ✓	5

2. It was not easy to depict all necessary elements of the reference system into the control structure diagram.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1 ✓	2	3	4	5

3. I found that the constructs of the modeling language allowed me to create all elements of the feedback control process model.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4 ✓	5

4. I could not figure out how to specify the inadequate control actions into the controller element of the feedback control process model.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3 ✓	4	5

5. I found it quite simple to define and to update the process model of the controller in the feedback control processes model.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2 ✓	3	4	5

Figure A.19: User 05 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs.

6. I could not find a type of generic control flaw that I was looking for in order to define the flaw in the feedback control process.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1 ✓	2	3	4	5
-----	---	---	---	---

7. I found that I was able to define the concept of a hazard and its causal factor elements without any difficulty.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	4 ✓	5
---	---	---	-----	---

8. While creating the hazard model, I could not find out how to specify the warning signs associated to each causal factor.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2 ✓	3	4	5
---	-----	---	---	---

9. Overall, I found that the constructs of the modeling languages enables me to complete all the tasks of the hazard and early warning analysis.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	4	5 ✓
---	---	---	---	-----

10. I found the the graphical icons representing the constructs of the modeling language are not appropriate.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2 ✓	3	4	5
---	-----	---	---	---

Figure A.20: User 05 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs (Page 2).

The Usability Evaluation Questionnaire

For each statements below circle the rating that indicates your assessment on the use of prototype software editor for the hazard and early warning sign analysis.

1. I found the prototype software editor to be user friendly.

STRONGLY DISAGREE DISAGREE NEUTRAL AGREE STRONGLY AGREE

1	2	3	4	5
---	---	---	---	---

2. It was not easy to use the prototype software editor.

STRONGLY DISAGREE DISAGREE NEUTRAL AGREE STRONGLY AGREE

1	2	3	4	5
---	---	---	---	---

3. I found using of the prototype software editor for early warning sign analysis to be very useful.

STRONGLY DISAGREE DISAGREE NEUTRAL AGREE STRONGLY AGREE

1	2	3	4	5
---	---	---	---	---

4. Conducting hazard and early warning sign analysis with the prototype software editor does not saves time.

STRONGLY DISAGREE DISAGREE NEUTRAL AGREE STRONGLY AGREE

1	2	3	4	5
---	---	---	---	---

5. While using the prototype software editor, I found that I could reenter the information stored into the models quickly and easily.

STRONGLY DISAGREE DISAGREE NEUTRAL AGREE STRONGLY AGREE

1	2	3	4	5
---	---	---	---	---

Figure A.21: User 06 ratings of the SUS questionnaire on satisfaction aspect of prototype software editor.

6. I found the system very cumbersome to use.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
X	2	3	4	5

7. I feel that most non hazard experts can learn to use this tool quickly.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	X	3	4	5

8. I needed to learn a lot of things before I could get going with this system.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	X	5

9. I feel that I would like to conduct hazard and early warning sign analysis using the prototype software editor more frequently .

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	X	5

10. Overall, I found the prototype software tool is unnecessarily complex.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	X	3	4	5

Figure A.22: User 06 ratings of the SUS questionnaire on satisfaction aspect of prototype software editor (Page 2).

The Usability Evaluation Questionnaire - Effectiveness Aspect

For each statements below circle the rating that indicates your assessment on the use of hazard and early warning sign analysis DSMLs using the prototype software editor.

1. I was able to define a control structure diagram model of the reference system using the constructs of the modeling language.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

2. It was not easy to depict all necessary elements of the reference system into the control structure diagram.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

3. I found that the constructs of the modeling language allowed me to create all elements of the feedback control process model.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

4. I could not figure out how to specify the inadequate control actions into the controller element of the feedback control process model.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

5. I found it quite simple to define and to update the process model of the controller in the feedback control processes model.

STRONGLY DISAGREE	DISAGREE	NEUTRAL	AGREE	STRONGLY AGREE
1	2	3	4	5

Figure A.23: User 06 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs.

6. I could not find a type of generic control flaw that I was looking for in order to define the flaw in the feedback control process.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	4	5
---	---	---	---	---

7. I found that I was able to define the concept of a hazard and its causal factor elements without any difficulty.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	4	5
---	---	---	---	---

8. While creating the hazard model, I could not find out how to specify the warning signs associated to each causal factor.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	4	5
---	---	---	---	---

9. Overall, I found that the constructs of the modeling languages enables me to complete all the tasks of the hazard and early warning analysis.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	4	5
---	---	---	---	---

10. I found the the graphical icons representing the constructs of the modeling language are not appropriate.

STRONGLY DISAGREE **DISAGREE** **NEUTRAL** **AGREE** **STRONGLY AGREE**

1	2	3	4	5
---	---	---	---	---

Figure A.24: User 06 ratings of the SUS questionnaire on the effectiveness aspect of the three hazard and early warning analysis DSMLs (Page 2).

- Akilli, G. (2005). User satisfaction evaluation of an educational website, *The Turkish Online Journal of Educational Technology-TOJET* 4(1): 85–92.
- Ansorg, R. and Schwabe, L. (2010). Domain-specific modeling as a pragmatic approach to neuronal model descriptions, *Brain Informatics* pp. 168–179.
- Artan, G., Restrepo, M., Asante, K. and Verdin, J. (2002). A flood early warning system for southern africa, *Proc., Pecora 15 and Land Satellite Information 4th Conf.*
- Atkinson, C. and Kuhne, T. (2003). Model-driven development: a metamodeling foundation, *Software, IEEE* 20(5): 36–41.
- Balagtas-Fernandez, F. and Hussmann, H. (2009). Applying domain-specific modeling to mobile health monitoring applications, *Information Technology: New Generations, 2009. ITNG'09. Sixth International Conference on*, IEEE, pp. 1682–1683.
- Bangor, A., Kortum, P. and Miller, J. (2009). Determining what individual sus scores mean: Adding an adjective rating scale, *Journal of usability studies* 4(3): 114–123.
- Bauer, C. and King, G. (2006). *Java Persistence with Hibernate*, Dreamtech Press.
- Beatty, W. and Kelley, D. (1994). Demonstration of a benchmarking technique to compare graduate education level of air force project managers and selected benchmarking partners, *Technical report*, DTIC Document.
- Bencomo, N. (2008). *Supporting the Modelling and Generation of Reflective Middleware Families and Applications using Dynamic Variability*, PhD Thesis, PhD thesis, Computing Department, Lancaster University, Lancaster, United Kingdom.
- Bencomo, N., Grace, P., Flores, C., Hughes, D. and Blair, G. (2008a). Genie - supporting the model driven development of reflective, component-based adaptive systems, pp. 811–814.
- Bencomo, N., Grace, P., Flores, C., Hughes, D. and Blair, G. (2008b). Genie - supporting the model driven development of reflective, component-based adaptive systems, pp. 811–814.

- Bencomo, N., Sawyer, P., Blair, G. and Grace, P. (2008). Dynamically adaptive systems are product lines too: Using model-driven techniques to capture dynamic variability of adaptive systems, *2nd International Workshop on Dynamic Software Product Lines (DSPL 2008), Limerick, Ireland*, Vol. 38, p. 40.
- Bevan, N. (1995). Measuring usability as quality of use, *Software Quality Journal* **4**(2): 115–130.
- Beydeda, S., Book, M. and Gruhn, V. (2005). *Model-driven software development*, Springer Verlag.
- Bézivin, J., Hillairet, G., Jouault, F., Kurtev, I. and Piers, W. (2005). Bridging the ms/dsl tools and the eclipse modeling framework, *Proceedings of the International Workshop on Software Factories at OOPSLA*.
- Blom, H., Bakker, G., Blanker, P., Daams, J., Everdij, M. and Klompstra, M. (2001). Accident risk assessment for advanced air traffic management, *Progress in Astronautics and Aeronautics* **193**: 463–480.
- Bogan, C. and English, M. (1994). Benchmarking for best practice: Winning through innovative adaptation, *New York*.
- Booch, G., Rumbaugh, J. and Jacobson, I. (1999). The unified modeling language user guide, *Addison-Wesley*.
- Boutekkouk, F., Benmohammed, M., Bilavarn, S., Auguin, M. et al. (2009). Uml2. 0 profiles for embedded systems and systems on a chip (socs), *JOT (Journal of Object Technology)* **8**(1): 135–157.
- Brooke, J. (1996). Sus-a quick and dirty usability scale, *Usability evaluation in industry* pp. 189–194.
- Brosnan, T. (1999). Early warning monitoring to detect hazardous events in water supplies, *ILSI Risk Science Institute Workshop Report*.
- Budinsky, F. (2004). *Eclipse modeling framework: a developer's guide*, Addison-Wesley Professional.
- Budinsky, F. and Brodsky, S. (2003). Merks, eclipse modeling framework.
- Budinsky, F., Steinberg, D., Merks, E., Ellersick, R. and Grose, T. (2003). *Eclipse Modeling Framework*, Addison Wesley Professional.

- Burgess, M. (2004). Fault tree creation and analysis tool: user manual, Published online at <http://www.iu.hio.no/FaultCat>, last accessed 11th October, 2012.
- Bussiere, M. and Fratzscher, M. (2006). Towards a new early warning system of financial crises, *Journal of International Money and Finance* **25**(6): 953–973.
- Cancila, D., Terrier, F., Belmonte, F., Dubois, H., Espinoza, H., Gérard, S. and Cuccuru, A. (2009a). Sophia: a modeling language for model-based safety engineering, *MoDELS*, Vol. 9, pp. 11–25.
- Cancila, D., Terrier, F., Belmonte, F., Dubois, H., Espinoza, H., Gérard, S. and Cuccuru, A. (2009b). Sophia a modeling language for model-based safety engineering.
- Carminati, B., Ferrari, E. and Bertino, E. (2005). Securing xml data in third-party distribution systems, *Proceedings of the 14th ACM international conference on Information and knowledge management*, ACM, pp. 99–106.
- Chen, Z. and Marx, D. (2005). Experiences with eclipse ide in programming courses, *Journal of Computing Sciences in Colleges* **21**(2): 104–112.
- Chin, J., Diehl, V. and Norman, K. (1988). Development of an instrument measuring user satisfaction of the human-computer interface, *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM, pp. 213–218.
- Clark, T., Evans, A., Sammut, P. and Willans, J. (2004). An executable meta-modelling facility for domain specific language design.
- Claypool, D., McNevin, T., Liu, W. and McNeill, K. (2009). Automated software defined radio deployment using domain specific modeling languages, *Mobile WiMAX Symposium, 2009. MWS'09. IEEE*, IEEE, pp. 157–162.
- Cohen, S., Hess, J., Kang, K., Novak, W. and Peteron, A. (1990). Feature-oriented domain analysis (foda) feasibility study, *Technical Report CMU/SEI-90-TR-21*, Software Engineering Institute, Carnegie Mellon University.
- Colombo, P., Lavazza, L., Coen-Porisini, A. and del Bianco, V. (2009). Towards a meta-model for problem frames: Conceptual issues and tool building

- support, *Software Engineering Advances, 2009. ICSEA'09. Fourth International Conference on*, IEEE, pp. 339–345.
- Cook, S., Jones, G., Kent, S. and Wills, A. (2007). *Domain-specific development with visual studio dsl tools*, Addison-Wesley Professional.
- Cooperation, R. (2003). . fmea software.
- Cuadrado, J. and Molina, J. (2007). Building domain-specific languages for model-driven development, *IEEE software* **24**(5): 48–55.
- Dehlinger, J. and Lutz, R. (2006). Plfaultcat: A product-line software fault tree analysis tool, *Automated Software Engineering* **13**(1): 169–193.
- Demir, A. (2006). Comparison of model-driven architecture and software factories in the con-text of model-driven development, *Model-Based Development of Computer-Based Systems and Model-Based Methodologies for Pervasive and Embedded Software, 2006. MBD/MOMPES 2006. Fourth and Third International Workshop on*, Ieee, pp. 9–pp.
- Di Ruscio, D., Lämmel, R. and Pierantonio, A. (2011). Automated co-evolution of gmf editor models, *Software Language Engineering* pp. 143–162.
- Dokas, I., Feehan, J. and Imran, S. (2012). Ewasap: An early warning sign identification approach based on stpa.
- Dokas, I., Karras, D. and Panagiotakopoulos, D. (2009). Fault tree analysis and fuzzy expert systems: Early warning and emergency response of landfill operations, *Environ. Model. Softw.* **24**(1): 8–25.
- Dokas, I., Wallace, R., Marinescu, R., Imran, S. and Foping, F. (2009). Towards a novel early warning service for state agencies: A feasibility study, *Information Technologies in Environmental Engineering* pp. 162–175.
- Dulac, N. and Leveson, N. (2004). An approach to design for safety in complex systems, *Int. Symposium on Systems Engineering (INCOSE)*.
- Dybvig, R. (2003). *The SCHEME programming language*, The MIT Press.
- Ellis, C. and Sun, C. (1998). Operational transformation in real-time group editors: issues, algorithms, and achievements, *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, Citeseer, pp. 59–68.

- Erdik, M., Fahjan, Y., Ozel, O., Alcik, H., Mert, A. and Gul, M. (2003). Istanbul earthquake rapid response and the early warning system, *Bulletin of Earthquake Engineering* **1**(1): 157–163.
- Espinoza, H., Cancila, D., Selic, B. and Gérard, S. (2009). Challenges in combining sysml and marte for model-based design of embedded systems, *Model Driven Architecture-Foundations and Applications*, Springer, pp. 98–113.
- Esser, R. and Janneck, J. (2001). A framework for defining domain-specific visual languages, *Workshop on Domain Specific Visual Languages, ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA-2001)*.
- Ewert, J., Guffanti, M., Murray, T. and (US), G. S. (2005). *An Assessment of Volcanic Threat and Monitoring Capabilities in the United States: Framework for a National Volcano Early Warning System NVEWS*, US Geological Survey.
- for Software Integrated Systems, I. (2005). Gme 5 users manual (v5.0), Vanderbilt University, at: <http://w3.isis.vanderbilt.edu/Projects/gme/GMEUMan.pdf>, last accessed 11th October, 2012.
- Frakes, W., Prieto-, Diaz, R. and Fox, C. (1998). Dare: Domain analysis and reuse environment, *Annals of Software Engineering* **5**(1): 125–141.
- Friedenthal, S., Moore, A. and Steiner, R. (2011). *A practical guide to SysML: the systems modeling language*, Morgan Kaufmann.
- Furtado, A. and Santos, A. (2006). Using domain-specific modeling towards computer games development industrialization, *6th OOPSLA Workshop on Domain-Specific Modeling (DSM'06)*.
- Gamma, E. and Beck, K. (2004). *Contributing to Eclipse: principles, patterns, and plug-ins*, Addison-Wesley Professional.
- García-Magariño, I., Fuentes-Fernández, R. and Gómez-Sanz, J. (2009). Guideline for the definition of emf metamodels using an entity-relationship approach, *Information and Software Technology* **51**(8): 1217–1230.
- Giachetti, G., Marín, B. and Pastor, O. (2009). Using uml as a domain-specific modeling language: A proposal for automatic generation of uml profiles, *Advanced Information Systems Engineering*, Springer, pp. 110–124.

- Greenfield, J. and Short, K. (2003). Software factories: assembling applications with patterns, models, frameworks and tools, *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, ACM, pp. 16–27.
- Greenfield, J. and Short, K. (2004). Software factories: Assembling applications with patterns, frameworks, models and tools.
- Hailpern, B. and Tarr, P. (2006). Model-driven development: The good, the bad, and the ugly, *IBM systems journal* **45**(3): 451–461.
- Heinrich, H. (1931). *Industrial accident prevention*, McGraw-Hill, New York.
- Hollnagel, E. (2004). *Barriers and accident prevention*, Ashgate Pub Ltd.
- Holzner, S. (2004). *Eclipse: A Java Developer's Guide*, O'Reilly & Associates, Inc.
- Hsu, K., Gupta, H. and Sorooshian, S. (1995). Artificial neural network modeling of the rainfall-runoff process, *Water resources research* **31**(10): 2517–2530.
- Hudson, R. (2003). Create an eclipse-based application using the graphical editing framework, *IBM developerWorks*, July.
- Hudson, R. and Shah, P. (2005). Gef in depth, tutorial slides.
- Imran, S., Dokas, I., Feehan, J. and Foping, F. (2010). A new application of domain specific modeling towards implementing an early warning service.
- Imran, S., Foping, F., Feehan, J. and Dokas, I. (2010a). Domain specific modeling language for early warning system: Using idef0 for domain analysis, *International Journal of Computer Science Issues(IJCSI)* **7**(5): 10–17.
- Imran, S., Foping, F., Feehan, J. and Dokas, I. (2010b). Domain specific modeling language for early warning system: Using idef0 for domain analysis, *IJCSI International Journal of Computer Science Issues* **7**(5): 1694–0814.
- Ishimatsu, T., Leveson, N., Thomas, J., Katahira, M., Miyamoto, Y. and Nakao, H. (2010). Modeling and hazard analysis using stpa, *Proceedings of the International Association for the Advancement of Space Safety Conference, Huntsville, Alabama (May 2010)*.

- ISO, W. (1998). 9241-11. ergonomic requirements for office work with visual display terminals (vdts), *The international organization for standardization*.
- Jensen, F. (1996). *An introduction to Bayesian networks*, Vol. 36, UCL press London.
- Johnson, K. M., Ratnayaka, D. D. and Brandt, M. J. (2009). *Twort's Water Supply*, Butterworth-Heinemann.
- Jouault, F. and Kurtev, I. (2006). On the architectural alignment of atl and qvt, *ACM symposium on Applied computing*, ACM.
- Karsai, G., Krahm, H., Pinkernell, C., Rumpe, B., Schindler, M. and Völkel, S. (2009). Design guidelines for domain specific languages, *The 9th OOP-SLA workshop on domain-specific modeling*, Citeseer.
- Kasemir, K., Chen, X., Purcell, J. and Danilova, K. (2011). Sns online display technologies for epics.
- Kelly, S. (2004). Comparison of eclipse emf/gef and metaedit+ for dsm, *19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, Workshop on Best Practices for Model Driven Software Development*.
- Kelly, S., Lyytinen, K. and Rossi, M. (1996a). Metaedit+ a fully configurable multi-user and multi-tool case and came environment, *Advanced Information Systems Engineering*, Springer, pp. 1–21.
- Kelly, S., Lyytinen, K. and Rossi, M. (1996b). Metaedit+ a fully configurable multi-user and multi-tool case and came environment, *Advanced Information Systems Engineering*, Springer, pp. 1–21.
- Kelly, S. and Pohjonen, R. (2009). Worst practices for domain-specific modeling, *Software, IEEE* **26**(4): 22–29.
- Kelly, S. and Tolvanen, J. (2008). *Domain-specific modeling: enabling full code generation*, Wiley-IEEE Computer Society Press.
- Kent, S. (2011). Is model driven development feasible?, MDSN Blogs, at http://blogs.msdn.com/b/stuart_kent/archive/2011/04/07/is-model-driven-development-feasible.aspx, last accessed 11th October, 2012.

- Kirakowski, J. and Corbett, M. (1993). Sumi: The software usability measurement inventory, *British journal of educational technology* **24**(3): 210–212.
- Kleppe, A. (2007). A language description is more than a metamodel, *Fourth International Workshop on Software Language Engineering, Nashville*.
- Kolovos, D., Paige, R. and Polack, F. (2008). The epsilon transformation language, *1st International Conference on Model Transformation, Zurich*.
- Kolovos, D., Rose, L. and Paige, R. (2012). The epsilon book.
- Krahn, H., Rumpe, B. and Völkel, S. (2007). Integrated definition of abstract and concrete syntax for textual languages, *Model Driven Engineering Languages and Systems* **4735**: 286–300.
- Lange, F. (2008). *Eclipse Rich Ajax Platform: Bringing Rich Client to the Web*, Springer.
- Lankford, W. (2000). Benchmarking: Understanding the basics, *Coastal Business Journal* **1**(1): 57–62.
- Lédeczi, Á., Bakay, A., Maroti, M., Volgyesi, P., Nordstrom, G., Sprinkle, J. and Karsai, G. (2001). Composing domain-specific design environments, *Computer* **34**(11): 44–51.
- Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., Thomason, C., Nordstrom, G., Sprinkle, J. and Volgyesi, P. (2001). The generic modeling environment, *Workshop on Intelligent Signal Processing, Budapest, Hungary*.
- Ledeczi, A., Nordstrom, G., Karsai, G., Volgyesi, P. and Maroti, M. (2001). On metamodel composition, *IEEE International Conference on Control Applications, 2001, (CCA'01)*, IEEE, pp. 756–760.
- Leveson, N. (1995). *Safeware: system safety and computers*, ACM.
- Leveson, N. (2002a). A new approach to system safety engineering, *Manuscript in preparation, draft can be viewed at <http://sunnyday.mit.edu/book2.pdf>*.
- Leveson, N. (2002b). System safety engineering: Back to the future, *Massachusetts Institute of Technology*.
- Leveson, N. (2004a). Model-based analysis of socio-technical risk, *Massachusetts Institute of Technology, Cambridge, MA, USA, Tech. Rep. ESD-WP-2004-08*.

- Leveson, N. (2004b). A new accident model for engineering safer systems, *Safety Science* **42**(4): 237–270.
- Leveson, N. (2004c). A systems theoretic approach to safety in software intensive systems, *Dependable and Secure Computing, IEEE Transactions on* **1**(1): 66–86.
- Leveson, N. (2012). *Engineering a safer world: Systems thinking applied to safety*, MIT Press (MA).
- Leveson, N., Daouk, M., Dulac, N. and Marais, K. (2003). Applying stamp in accident analysis, *NASA CONFERENCE PUBLICATION*, NASA; 1998, pp. 177–198.
- Leveson, N., Dulac, N., Zipkin, D., Cutcher-Gershenfeld, J., Carroll, J. and Barrett, B. (2006). Engineering resilience into safety-critical systems, *Resilience Engineering–Concepts and Precepts*. Ashgate Aldershot pp. 95–123.
- Lewis, J. and Launchbury, J. (1998). Initial suite of small language definitions and implementations for dsdl, *Technical report*, DTIC Document.
- MARTE, O. (2008). A uml profile for marte: Modeling and analysis of real-time embedded systems, beta 2, ptc/2008-06-09.
- Mattsson, A., Lundell, B., Lings, B. and Fitzgerald, B. (2009). Linking model-driven development and software architecture: A case study, *Software Engineering, IEEE Transactions* **35**(1): 83–93.
- Mernik, M., Heering, J. and Sloane, A. (2005). When and how to develop domain-specific languages, *ACM Computing Surveys (CSUR)* **37**(4): 316–344.
- Miller, J., Mukerji, J. et al. (2003). Mda guide version 1.0. 1, *Object Management Group* **234**: 51.
- Miotto, E. and Vardanega, T. (2009). On the integration of domain-specific and scientific bodies of knowledge in model driven engineering, *In: Procs. of STANDRTS'09, Dublin, Ireland*.
- Moore, B., Organization, I. B. M. C. I. T. S. and (Firme), S. B. O. (2004). *Eclipse development using the graphical editing framework and the eclipse modeling framework*, IBM, International Technical Support Organization.

- Nordstrom, G., Sztipanovits, J., Karsai, G. and Ledeczi, A. (1999). Metamodeling-rapid design and evolution of domain-specific modeling environments, *Proceedings of the IEEE ECBS'99 Conference*, IEEE, pp. 68–74.
- Northover, S. and Wilson, M. (2004). *Swt: the standard widget toolkit*, Addison-Wesley.
- Oberortner, E., Zdun, U. and Dustdar, S. (2009). Tailoring a model-driven quality-of-service dsl for various stakeholders, *Modeling in Software Engineering, 2009. MISE'09. ICSE Workshop on*, IEEE, pp. 20–25.
- Olumofin, F. and Mišić, V. (2006). Preserving architectural knowledge through domain-specific modeling, *6th OOPSLA Workshop on Domain-Specific Modeling (DSM'06)*, pp. 98–104.
- OMG, M. (2005). Qvt final adopted specification.
- Pahl, C. and Barrett, R. (2007). Semantic model-driven development of service-centric software architectures, *Software Engineering Methods for Service-Oriented Architecture 2007 (SEMSEA 2007)*.
- Pitkänen, R. and Mikkonen, T. (2006). Lightweight domain-specific modeling and model-driven development, *OOPSLA 6th Workshop on Domain Specific Modeling*, pp. 159–168.
- Prieto-Díaz, R. (1990). Domain analysis: an introduction, *ACM SIGSOFT Software Engineering Notes* **15**(2): 47–54.
- Qureshi, Z. (2008). A review of accident modelling approaches for complex critical sociotechnical systems, *Technical report*, DTIC Document.
- Rabiner, L. and Juang, B. (1986). An introduction to hidden markov models, *ASSP Magazine, IEEE* **3**(1): 4–16.
- Rasmussen, J. and Svedung, I. (2000). *Proactive risk management in a dynamic society*, Swedish Rescue Services Agency Karlstad, Sweden.
- Ráth, I., Ökrös, A. and Varró, D. (2010). Synchronization of abstract and concrete syntax in domain-specific modeling languages, *Software and Systems Modeling* **9**(4): 453–471.
- Rath, I. and Varro, D. (2006). *Declarative specification of domain specific visual languages*, PhD thesis, Master's thesis, Budapest University of Technology and Economics.

- Reason, J. and Reason, J. (1997). *Managing the risks of organizational accidents*, Vol. 6, Ashgate Aldershot.
- Rivera, J., Durán, F. and Vallecillo, A. (2009). Formal specification and analysis of domain specific models using maude, *Simulation* **85**(11-12): 778–792.
- Robert, S., Gérard, S., Terrier, F. and Lagarde, F. (2009). A lightweight approach for domain-specific modeling languages design, *Software Engineering and Advanced Applications, 2009. SEAA'09. 35th Eu-romicro Conference on*, IEEE, pp. 155–161.
- Rudloff, A., Lauterjung, J., Münch, U. and Tinti, S. (2009). The gitews project (german-indonesian tsunami early warning system), *Nat. Hazards Earth Syst. Sci* **9**: 1381–1382.
- Sadilek, D. (2007a). Prototyping and simulating domain-specific languages for wireless sensor networks.
- Sadilek, D. (2007b). Prototyping domain-specific languages for wireless sensor networks, *Proc. of the 4th Int. Workshop on Software Language Engineering*, pp. 76–91.
- Salvendy, G. (1997). *Handbook of human factors and ergonomics*, John Wiley & Sons New York, NY.
- Sauro, J. and Lewis, J. (2011). When designing usability questionnaires, does it hurt to be positive?, *Proceedings of the 2011 annual conference on Human factors in computing systems*, ACM, pp. 2215–2224.
- Schauerhuber, A., Wimmer, M. and Kapsammer, E. (2006). Bridging existing web modeling languages to model-driven engineering: a metamodel for webml, *Workshop proceedings of the sixth international conference on Web engineering*, ACM, p. 5.
- Selic, B. (2003). The pragmatics of model-driven development, *Software, IEEE* **20**(5): 19–25.
- Selic, B. (2006). Model-driven development: Its essence and opportunities, *Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC 2006)*, IEEE Computer Society, pp. 313–319.

- Selic, B. (2007). A systematic approach to domain-specific language design using uml, *Object and Component-Oriented Real-Time Distributed Computing, 2007. ISORC'07. 10th IEEE International Symposium on*, Ieee, pp. 2–9.
- Selic, B. (2011). The theory and practice of modeling language design for model-based software engineering-a personal perspective, *Generative and Transformational Techniques in Software Engineering III* pp. 290–321.
- Shikhar, P. (2008). Recommendations for performance benchmarking, *White Paper, Infosys - Building tomorrows enterprise* .
- Silingas, D., Vitiutinas, R., Armonas, A. and Nemuraite, L. (2009). Domain-specific modeling environment based on uml profiles, *Targamadze, A., Butleris, R., Butkiene, R.(eds.), Information Technologies* pp. 167–177.
- Sivonen, S. (2008). Domain-specific modelling language and code generator for developing repository-based eclipse plug-ins.
- Sommerville, I. (2011). *Software Engineering - Ninth Edition*, Addison-Wesley.
- Spinellis, D. (2001). Notable design patterns for domain-specific languages, *Journal of Systems and Software* **56**(1): 91–99.
- Spivey, J. (1992). *The Z notation: a reference manual*, Prentice Hall International (UK) Ltd.
- Stahl, T., Völter, M. and Czarnecki, K. (2006). *Model-driven software development: technology, engineering, management*, John Wiley & Sons.
- Steinberg, D., Budinsky, F., Paternostro, M. and Merks, E. (2009). Emf: Eclipse modeling framework (eclipse).
- Stringfellow, M. (2011). *Accident analysis and hazard analysis for human and organizational factors*, PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics.
- Stringfellow, M., Leveson, N. and Owens, B. (2010). Safety-driven design for software-intensive aerospace and automotive systems, *Proceedings of the IEEE* **98**(4): 515–525.

- Taentzer, G. (2006). Towards generating domain-specific model editors with complex editing commands, *Proc. International Workshop Eclipse Technology eXchange (eTX), Satellite Event of ECOOP*.
- Tairas, R., Mernik, M. and Gray, J. (2009). Using ontologies in the domain analysis of domain-specific languages, *Models in Software Engineering* pp. 332–342.
- team, S. (2008). Omg systems modeling language(sysml)version 1.1.
- Thomas, D. (2004). Mda: Revenge of the modelers or uml utopia?, *Software, IEEE* **21**(3): 15–17.
- Tolvanen, J. (2004). Metaedit+: domain-specific modeling for full code generation demonstrated [gpce], *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, ACM, pp. 39–40.
- Tolvanen, J., Pohjonen, R. and Kelly, S. (2007). Advanced tooling for domain-specific modeling: Metaedit+, *7th OOPSLA workshop on Domain-Specific Modeling*.
- Tranoris, C. and Denazis, S. (2010). Federation computing: A pragmatic approach for the future internet, *Network and Service Management (CNSM), 2010 International Conference on*, IEEE, pp. 190–197.
- Tung, W., Quek, C. and Cheng, P. (2004). Genso-ews: a novel neural-fuzzy based early warning system for predicting bank failures, *Neural Networks* **17**(4): 567–587.
- UML, O. (2003). 2.0 infrastructure specification. object management group.
- UN/ISDR (2006). Developing early warning systems: A checklist,, *Third International Conference on Early Warning, From concept to action (EWC III)*.
- Valerio, A., Succi, G. and Fenaroli, M. (1997). Domain analysis and framework-based software development, *ACM SIGAPP Applied Computing Review* **5**(2): 4–15.
- Van Deursen, A., Klint, P. and Visser, J. (2000). Domain-specific languages: An annotated bibliography, *ACM Sigplan Notices* **35**(6): 26–36.
- Van Rossum, G. and Drake, F. (2003). *Python language reference manual*, Network Theory Limited.

- Vidal, J., De Lamotte, F., Gogniat, G., Soulard, P. and Diguët, J. (2009). A co-design approach for embedded system modeling and code generation with uml and marte, *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE'09.*, IEEE, pp. 226–231.
- Vuolle, M., Aula, A., Kulju, M., Vainio, T. and Wigelius, H. (2008). Identifying usability and productivity dimensions for measuring the success of mobile business services, *Advances in Human-Computer Interaction* **2008**.
- Weiss, D. and Lai, C. (1999). Software product-line engineering: A family-based software development process author: David m. weiss, chi tau robert lai.
- WHO (2006). Guidelines for disease surveillance/early warning and response - middle east crisis, *Communicable Diseases Working Group on Emergencies, World Health Organization WHO/CDS/NTD/DCE/2006.6*.
- Williams, S. and Kindel, C. (1994). The component object model: A technical overview, *Technical report*, Microsoft Technical Report.
- Woocher, L. (2006). Developing a strategy, methods and tools for genocide early warning, *Report prepared for the Office of the Special Adviser to the UN Secretary General on the Prevention of Genocide, Center for International Conflict Resolution, Columbia University, New York*.
- Woods, D. (2009). Escaping failures of foresight, *Safety science* **47**(4): 498–501.
- Yet-Pole, I. (2003). Development and applications of casehat—a multipurpose computer aided hazard analysis automation system used in semiconductor manufacturing industry, *Journal of Loss Prevention in the Process Industries* **16**(4): 271–279.
- Zan, L., Latini, G., Piscina, E., Polloni, G. and Baldelli, P. (2002). Landslides early warning monitoring system, *Geoscience and Remote Sensing Symposium, 2002. IGARSS'02. 2002 IEEE International*, Vol. 1, IEEE, pp. 188–190.
- Zdun, U. (2002). Domain-specifically tailorable languages and software architectures, *University of Essen, Germany*.
- Zenko, M. and Friedman, R. (2011). Un early warning for preventing conflict, *International Peacekeeping* **18**(1): 21–37.

- Zisman, A. (2000). An overview of xml, *Computing & Control Engineering Journal* **11**(4): 165–167.
- Zschau, J. and Küppers, A. (2003). *Early warning systems for natural disaster reduction*, Springer Verlag.
- Zviran, M., Glezer, C. and Avni, I. (2006). User satisfaction from commercial web sites: The effect of design and use, *Information & Management* **43**(2): 157–178.