Master Thesis
Software Engineering
Thesis no: MSE-2007:07
January 2007

Comparison of Microsoft DSL Tools and Eclipse Modeling Frameworks for Domain-Specific Modeling In the context of the Model-Driven Development

Turhan Özgür

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

This thesis is submitted to the School of Engineering at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author(s):
Turhan Özgür
E-mail: turhanozgur@gmail.com

University advisor(s):
Dr. Göran Fries
Department of Computer Science and Software Engineering

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet : www.bth.se/tek
Phone : +46 457 38 50 00
Fax : + 46 457 271 25

ACKNOWLEDGEMENT

I am thankful to my parents, for their support and encouragement to make it possible for me to pursue my academic studies. Without their support, it would not be possible for me to write this master thesis.

I am thankful to all Swedish tax-payers who are in favour of sharing their modern educational and cultural values by providing international study environments like Blekinge Institute of Technology. Their contributions to the citizens of developing countries will eventually contribute to the economical and political stability of the world and increase tolerance among people from distinct cultures and ethnicities.

ABSTRACT

Today it is realized by industry that automation of software development leads to increased productivity, maintainability and higher quality. Model-Driven Development (MDD) aims to replace manual software development methods by automated methods using Domain-Specific Languages (DSLs) to express domain concepts effectively. Main actors in software industry, Microsoft and IBM have recognized the need to provide technologies and tools to allow building DSLs to support MDD. On the one hand, Microsoft is building DSL Tools integrated in Visual Studio 2005; on the other hand IBM is contributing to the development of Eclipse Modeling Frameworks (EMF/GEF/GMF), both tools aim to make development and deployment of DSLs easier. Software practitioners seek for guidelines regarding how to adopt these tools. In this thesis, the author presents the current state-of-the-art in MDD standards and Domain-Specific Modeling (DSM). Furthermore, the author presents current state-of-the-tools for DSM and performs a comparison of Microsoft DSL Tools and Eclipse EMF/GEF/GMF Frameworks based on a set of evaluation criteria. For the purpose of comparison the author developed two DSL designers (one by using each DSM tool). Based on the experiences gained in development of these DSL designers, the author prepared guidelines regarding how to adopt these tools to existing development environments as well as their advantages and drawbacks.

.

Keywords: Model-Driven Development, Domain-Specific Modeling, Domain-Specific Languages

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

The aim of this chapter is to introduce the study area and to provide the motivation for this thesis. It is structured as follows: Section 1.1 presents the problem description and background of the study area; Section 1.2 presents the author's motivation for this thesis; Section 1.3 presents the research objectives and posed research questions; Section 1.4 presents the selected research methodology for this thesis. Finally, Section 1.5 provides a brief overview of the chapters in this thesis.

1.1 Background

Today traditional software development is in search for more automation due to increasing demand for software from the market. The typical problems that rise from the complexity of software are:

- Inability to explicitly capture domain knowledge and practices and to make them available for reuse [1].
- Lack of domain knowledge among the developers results in unproductive development process [1].
- Incapability of domain experts to understand technology related issues involved in software development [1].
- Most of the software is still developed manually from scratch. As a result, software development is slow, expensive and defect-prone.

Model-Driven Engineering (MDE) technologies address the inability of third-generation languages to alleviate the complexity of platforms and express domain concepts effectively [54]. Model-Driven Development (MDD) which is an Object Management Group (OMG) trademark [18] is a software development approach where the first class entities are the models and the model transformations. The aim of the MDD is to raise the level of abstraction with tools and to provide higher level of automation. Two currently leading approaches to MDD are Model Driven Architecture (MDA) [23] and Software Factories [24]. MDA is the proposal of the OMG for supporting the MDD approach using its own standards such as Meta-Object Facility (MOF) [15], XML Metadata Interchange (XMI) [14], Unified Modeling Language (UML) [13], and Common Warehouse Metamodel (CWM) [16]. MDA is one realization of the broader vision of MDE [26]. On the other hand, Software Factories, as proposed by Microsoft, is a new software development paradigm, which is a development environment configured to support the rapid development of a specific type of application [1]. Software Factories focuses on product line development which is concerned with developing a set of similar but distinct products. Software Factories approach enables a high degree of reuse of existing assets and development of new reusable assets for a specific member of a product family [27]. Software Factories makes the use of Domain-Specific Modeling (DSM) by utilizing Domain-Specific Languages (DSLs) for modeling various aspects of a software system. A DSL uses concepts and terms of the problem domain and provides graphical and textual notations to create models [27].

Nowadays two major industrial platforms employ the MDE principles and standards: IBM's Eclipse Modeling Framework (EMF) [20] and Graphical Editing Framework (GEF) [21] employ MDA standards and Microsoft Domain-Specific Language Tools (MS/DSL Tools) employs Software Factories standards. EMF is a modeling framework and code generation facility for building tools and applications based on a structured data model [25]. GEF provides the graphical support needed for building a diagram editor on the top of the EMF framework. EMF and GEF together can be used to build a DSM support. In addition, Graphical Modeling Framework (GMF) [22] is the new generative framework for

developing graphical editors for providing support for DSM by leveraging EMF and GEF. [28]. On the other hand, MS/DSL Tools is a suite of tools for creating, editing, visualizing, and using domain-specific data for automating the enterprise software development process. Apart from EMF, MS/DSL Tools is not UML [13] based (neither MOF nor XMI) and it enables users to design graphical languages and to generate code and other artifacts from these languages [26]. In this sense, the Eclipse Modeling Frameworks and MS/DSL Tools offer different approaches to define support for DSM.

1.1.1 Abbreviations

Table 1 lists the abbreviations that are often used in this thesis:

Name/acronym	**Definition**
AMMA	ATLAS Model Management Architecture
API	Application Programming Interface
ATL	ATLAS Transformation Language
BEL	Business Entity Language
CTP	Community Technology Preview
CWM	Common Warehouse Metamodel
DSM	Domain-Specific Modeling
DSL	Domain-Specific Language
EMF	Eclipse Modeling Framework
ER	Entity Relationship
GEF	Graphical Editing Framework
GME	Generic Modeling Environment
GMF	Graphical Modeling Framework
GMT	Generative Modeling Technologies
GPL	General-purpose Language
IDE	Integrated Development Environment
KM3	Kernel MetaMetaModel
M2M	Model-to-Model Transformation
MDA	Model Driven Architecture
MDD	Model-Driven Development
MIC	Model Integrated Computing
MDE	Model-Driven Engineering
MOF	Meta-Object Facility
MS/DSL Tools	Microsoft Domain-Specific Language Tools
OCL	Object Constraint Language
OMG	Object Management Group
QVT	Query-View-Transformation
SDK	Software Development Kit
SPEM	Software Process Engineering Metamodel
UML	Unified Modeling Language
VIATRA	VIsual Automated model TRAnsformations
XMI	XML Metadata Interchange
XML	Extensible Markup Language

Table 1 Abbreviations

1.2 Motivation

Software industry has progressed through three waves so far as stated by Humphrey [70]:

- The first wave is characterized by Waterfall lifecycle and structured methods. It provided the ability to represent software activities diagrammatically. This wave brought software development nearer to engineering discipline.
- The second wave is the process maturity movement which aims to bring software development even nearer to becoming an engineering discipline. The scope of this wave is wider than analysis, design and coding. It covers all aspects of the software development process and its supporting activities.
- The third wave as described by Zahran [71] is software industrialization where software development turns into a manufacturing and assembly of components. Having a disciplined process is a prerequisite for the realization of software industrialization.

Model-Driven Development (MDD) is an emerging approach which can be used for realization of software industrialization. Many case studies from the industry showed that MDD provides marginal improvements in the productivity of software development and in the quality of the software products. MDD aims to replace manual software development methods by automated methods using Domain-Specific Languages (DSLs) to express domain concepts effectively. Two main actors in software industry, Microsoft and IBM are developing their own tools and technologies to facilitate building DSLs.

The main reason to choose MS/DSL Tools and Eclipse Modeling Frameworks (EMF/GEF/GMF) for comparison is their full support for Domain-Specific Modeling. Since Microsoft aims to include DSL Tools in the next release of Visual Studio 2005 Team System, it is likely that DSL Tools will gain high acceptance from developers who are already developing software using Microsoft Integrated Development Environments (IDEs). On the other hand, Eclipse is a fast-growing open source community, whose projects are focused on building extensible frameworks, tools and runtimes for building, deploying and managing software throughout the lifecycle. Its extensible characteristic of Eclipse facilitates building and integrating modeling frameworks to support DSL development.

1.3 Objectives and Questions

The purpose of this study is to perform a comparison of Microsoft DSL Tools and Eclipse Modeling Frameworks for Domain Specific Modeling in the context of Model-Driven Development. The research objectives of this thesis are as follows:

1. To perform literature search and review on MDD approaches (Software Factories and MDA), DSM and their application to MS/DSL Tools and Eclipse Modeling Frameworks;
2. To evaluate MS/DSL Tools and Eclipse Modeling Frameworks considering criteria such as metamodeling, graphical notation, model transformations, etc.;
3. To prepare guidelines regarding advantages and disadvantages of each DSM tool.

The research questions that this thesis aims to answer are:

1. What is the current state-of-the-art in MDD standards and DSM?
2. What are the advantages and disadvantages of MS/DSL Tools and Eclipse EMF/GEF/GMF for development of Domain-Specific Languages?
3. How can models defined under two different modeling technologies be exchanged?
4. How can DSM tools be integrated with existing development tools?

1.4 Methodology

This project is an evaluation project which is defined under the empirical software engineering. First, a literature survey of current MDD standards and DSM was performed. Later, two DSL designers (one by using MS/DSL Tools and one by using Eclipse Modeling Frameworks) for a small business entity language were developed. Finally, experiences gained in development of both DSL designers were compared based on a set of evaluation criteria and guidelines were prepared regarding advantages and drawbacks of each DSM tool.
This thesis does not focus on any particular domain for the comparison of selected DSM tools; rather it performs a comparison based on a set of evaluation criteria such as metamodeling, graphical notation, model transformations, etc. Selected DSM tools are independent of any particular domain; however in the future tool vendors may develop specific DSM tools for particular domains. This thesis does not aim to recommend a particular tool; rather it aims to provide guidelines to practitioners, so they can decide which DSM tool better addresses their project specific needs.

1.5 Overview

The aim of this section is to give a brief overview about the chapters in this thesis. Each chapter has different focus and scope.

Chapter 2 presents an overview of the current state-of-the-art in Model-Driven Engineering -which is a broader vision of Model-Driven Development- and discusses recognized benefits of MDE considering different organizational and quality attributes. It also discusses about current MDE standards such as Model-Driven Architecture and Software Factories and presents a comparison of these standards.

Chapter 3 presents an overview of the current state-of-the-art in Domain-Specific Modeling. It provides discussions about Domain-Specific Language development, its related benefits and risks and performs a comparison of Domain-Specific Languages and UML. Furthermore, it presents the current state of the Domain-Specific Modeling tools, which are selected for comparison, Microsoft DSL Tools and Eclipse Modeling Project including EMF/GEF/GMF.

Chapter 4 presents different phases in DSL development and discusses about how to integrate DSM tools to existing development environment. Later, the author developed two Domain-Specific Language designers by using both MS/DSL Tools and Eclipse EMF/GEF/GMF. Furthermore, an evaluation of two modeling tools was performed based on a set of criteria under guidance of experiences gained in development of two DSL designers (one by using each modeling tool).

Chapter 5 presents related work that has been done in the literature and discusses how these literature references correspond to the author's own thesis work.

Chapter 6 presents some discussions about the choice of evaluation and how the thesis work relates to the problem definition.

Chapter 7 presents the overall conclusions that are extracted from the thesis work and the development of DSLs using two DSM tools.

Chapter 8 discusses the future work that is relevant for both the author and other researches who are interested in continuing studies in the thesis area.

Chapter 9 presents the selected references comprised of a variety of information sources such as research articles, web resources and literature books.

2 MODEL DRIVEN ENGINEERING

The goal of this chapter is to provide an overview of current state-of-art in Model-Driven Engineering. It is structured as follows: Section 2.1 and 2.2 present the foundation, principles and standards behind Model-Driven Engineering; Section 2.3 presents benefits of Model-Driven Engineering from quality attributes and productivity perspectives; Section 2.4 and 2.5 present the major MDE approaches Software Factories and Model-Driven Architecture. Finally, Section 2.6 presents a comparison of these MDE approaches with respect to differences and similarities between them.

2.1 Introduction

Today the development of software systems is getting more and more complex and widely distributed. Thus, developers must have knowledge of a wide range of technologies. End users expect more fast, reliable and scalable results despite unpredictable changes in the market [29]. The software industry always looked for means to narrow the gap on the way from requirements to software. Many programming languages have been created and development paradigms were invented. High-level general purpose programming languages were created to facilitate building applications. The object oriented paradigm aims to move software closer to the reality that is assumed to be object-oriented [5]. The rapid growth of software across many fields increases the demand for software significantly. The demand currently exceeds the ability to produce software by a large margin and this software-gap is gradually increasing in time [50]. Recall the transition from Assembly language to C for productive application development. This transition was driven by businesses seeking competitive advantage by lowering cost and time to market, and raising product quality by using automation [1].

Today's expectation is the displacement of the current generation of manual software development methods by automated methods using higher-level languages over a period of many years [1]. To address this issue, the traditional development means need to be enhanced by new approaches which would raise the level of abstraction even more by bridging the software engineering discipline closer to the actual domain where it is applied [50].

In order to industrialize software development process similar to car or television manufacturing industries, the development of modeling languages which directly represent problem domain is needed. Instead of being focused on a particular technological problem such as programming or data interchange, these modeling languages will be used to build models of domains that they address [29]. In this sense, Model-Driven Development (MDD) which is an Object Management Group (OMG) trademark [18] has potential to become a solution to the software-gap problem. In order to realize MDD, the traditional development process would be gradually moved from writing code to creating domain-specific models and generating code and other artifacts from these models [50]. Domain specific languages are defined in terms of domain problems that should be solved by the software. In order to increase productivity in software development process and move requirements closer to the product, domain specific languages can be considered as assembly lines which are similar to the assembly lines in automobile industry [5].

Unlike general-purpose modeling languages such as OMG's Unified Modeling Language (UML) [13], Domain-Specific Languages (DSLs) brings the modeling much closer to the domain experts and enables easier maintenance and evolution of such models which contributes to the desired productivity increase and to the agility of MDD. However, MDD

must overcome the inactivity of existing approaches, and in addition to all required technical ingredients, the management support is needed to make the paradigm shift happen [50]. Furthermore, much practical and theoretical development needs to occur before MDD becomes a dominant style of development [51].

2.2 Principles, Standards and Tools

Dominant view today is that models may have value in early phases of development to sketch-out and communicate high-level design ideas meaning that models play only secondary, mostly documentation role. In contrast, models become essential artifacts in MDD rather than serving an inessential supporting purpose [51]. MDD employs models to represent a system's elements and their relationships. Models serve as input and output at all phases of system development until the system is generated [55].

There are two key themes in the core of MDD [51]:

- *Raising the level of abstraction* of specifications to be closer to the problem domain by using modeling languages with higher-level and better behaved constructs.
- *Raising the level of automation* by using technology to bridge the semantic gap between the specification (the model) and the implementation (the generated code).

The practical effects of these key themes are higher product quality and productivity increase in development. Before introducing MDD into an established production development, the costs of purchasing new tools, training staff and management should be considered carefully. In addition, the lack of available MDD experts and the human aspects such as staff resistance to change may be other difficulties. Besides, the investment in MDD continues to pay back in long-term after the initial implementation since the model is much easier to understand and maintain than code [51].

Development activities such as debugging, building and deploying are partially or fully automated today by using information captured by source code files and other implementation artifacts. MDD can provide more extensive automation of these activities by using information captured by models. The essential features of MDD are [1]:

- Using domain specific languages to write specifications of software that capture developer intent in computational forms.
- Isolating and encapsulating variability points.
- Using model driven environments to automatically generate partial or complete implementations from specifications.

Model-Driven Engineering (MDE) which is broader vision of MDD encompasses research trends related to generative and transformational techniques in software engineering, system engineering and data engineering [5]. MDE is a promising approach to address platform complexity and the inability of third-generation languages to alleviate this complexity and express domain concepts effectively. In this sense, MDE technologies combine DSLs and transformation engines and generators [54]. The basic principle of MDE is to consider any software artifact a model or a model element [5]. PlanetMDE [9] is a free and open web portal to everybody who wants to share knowledge about MDE. It is independent from any particular project or standardization body [9]. Therefore, it provides a variety of information source related to MDE as inputs to this thesis.

Nowadays major MDE approaches are OMG Model Driven Architecture (MDA) proposal [23], Microsoft Software Factories [24] and Model Integrated Computing (MIC) [41] shown in Figure 1 [5]. MDA is a realization of a set of OMG standards like Meta-Object Facility (MOF) [15], Common Warehouse Metamodel (CWM) [16], XML Metadata Interchange (XMI) [14], Object Constraint Language (OCL), Unified Modeling Language (UML) and

Software Process Engineering Metamodel (SPEM) [10]. MDA represents systems using UML (along with specific profiles) and transforms these models into artifacts that run on a variety of platforms such as Enterprise JavaBeans (EJB), Microsoft.NET and CORBA Component Model (CCM). Unlike MDA, MIC employs DSLs to represent system elements and their relationships as well as their transformations to platform-specific artifacts [55]. On the other hand, Software Factories, as proposed by Microsoft, is a new software development paradigm, which is a development environment configured to support the rapid development of a specific type of application. Software Factories makes the use of Domain-Specific Modeling (DSM) by utilizing DSLs for modeling various aspects of a software system [1]. While MDE takes its roots in the OMG MDA industrial standard, Microsoft and IBM develop their own technologies: Microsoft develops MS/DSL Tools employ Software Factories approach and IBM contributes to Eclipse Modeling Framework (EMF) [20], Graphical Editing Framework (GEF) [21] and Graphical Modeling Framework (GMF) [22] employ MDA approach [5]. Generic Modeling Environment (GME) is an open source, model-integrated design environment for creating DSLs and program synthesis [55]. It is developed by the Institute of Software Integrated Systems at Vanderbilt University and it employs MIC approach [45]. Eclipse modeling frameworks and MS/DSL Tools offer different approaches to defining support for Domain-Specific Modeling (DSM).

MDE tools impose domain-specific constraints and platform model checking that can detect errors early in the development life cycle [54]. An agile MDE environment should include a large number of small DSLs defined by well-focused metamodels and provide many tools based on metamodels. Interoperability between tools is provided via model transformations [5].

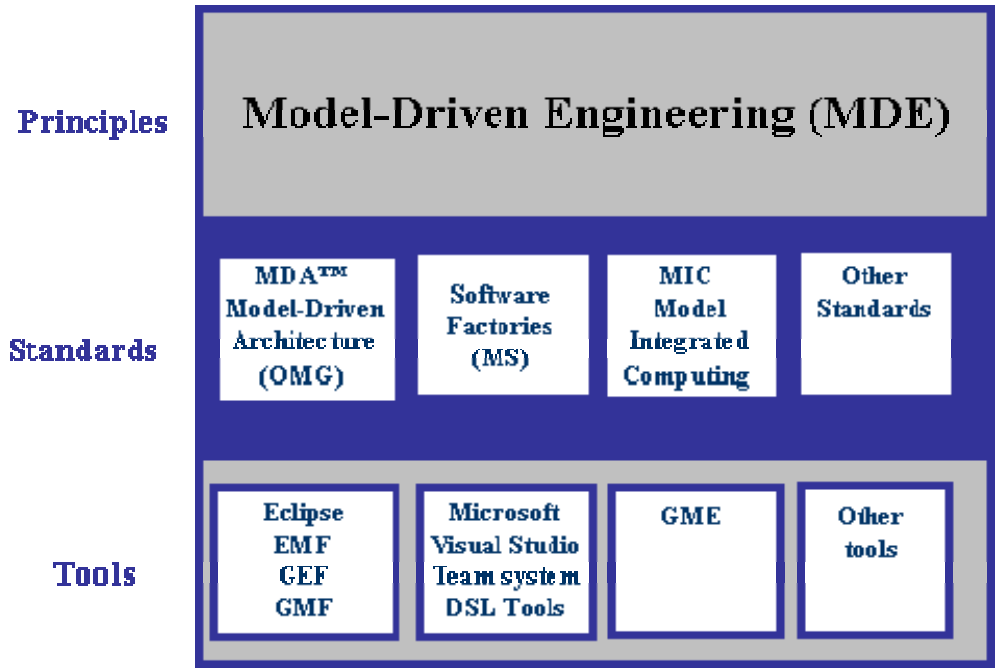

Figure 1 Model-Driven Engineering [26]

2.2.1 Models, Metamodeling

A *model* is an abstract description of software that hides information about some aspects of the software [1]. A deeper definition is provided by [6]: “A model is a simplification of a system built with an intended goal in mind. The model should be able to answer questions in the place of the actual system.” A model should be easier to use than the original system in such a way that many details from the source system are abstracted out, and only a few are implemented in the target model [6].

"A *meta-model* is the explicit specification of an abstraction. In order to define the abstraction, the meta-model identifies a list of relevant concepts and a list of relevant relationships between these concepts."[6]

A particular model can be extracted from a system by using a specific meta-model. Different models of the same system can be observed and manipulated, and each model can be represented by a different meta-model [6]. In this sense, metamodel acts as a filter which specifies the set of concerns that should be considered while creating a model [5].Favre [8] discusses that there are four groups of meanings for "model" word: model-as-representation, model-as-example, model-as-type and model-as-mold. Since development technologies become more and more complex, in order to increase productivity, modeling is necessary to bridge the gap between business and technology. In this sense, models allow the domain problems to be described by using terms that are familiar to domain experts rather than term that only familiar to IT experts [29].

Metamodeling identifies general concepts and their relations in a given problem domain and forms a modeling language used to create executable domain models. The executable model must represent target domain concepts where the software system is to be used. This allows domain experts not only validating satisfaction of their requirements, but also they learn to model the system themselves. A metamodel specifies the scope of what may be defined in a model, what elements may be contained in the model and how these elements relate to each other. Metamodel abstracts from specifics of implementation technologies, it focuses on defining domain concepts [5].

Metamodeling is used to create abstract syntax of a modeling language. The Meta Object Facility (MOF) [48] is the metamodeling technique standardized by OMG where the metamodel is written as a simplified UML Class Diagram and Object Constraint Language (OCL) is used to define constraints on the abstract syntax [47].
The layered architecture of UML has the following four layers [6]:

- M3: the meta-metamodel level (contains only the MOF).
- M2: the metamodel level (contains any kind of metamodel, including the UML metamodel).
- M1: the model level (any model with a corresponding metamodel from M2).
- M0: the concrete level (any real situation, unique in space and time, represented by a given model from M1).

Metamodeling significantly eases the implementation of DSLs and provides support for experimenting with the modeling language as it is built. Thus, metamodel-based language definition also assists in the task of constructing generators that reduce the burden of tool creation and maintenance [75].

Models do not state how the constructs they describe are implemented. Models can be used to separate concerns by isolating information about specific aspects of the software. In this way, they can be managed independently. Graphical visualization can make models easier to understand, however models do not have to be visualized graphically. Thus, the popularity of languages designed for graphical visualization such as UML should not lead to association of models with graphical visualization [1]. Besides, the simplicity of visualization can come from hiding important model details. An inappropriate visualization may give a misleading impression of the model. On the other hand, multiple visualizations of models can help to increase understandability of models. Multiple visualizations of models introduce new pitfalls: the expense of maintaining multiple model representations and possible inconsistencies between models. Thus, it is necessary to evaluate whether increase in understandability of models is worth the increased cost of development [62]. As an alternative to graphical visualizations, the information captured by a model can be rendered

as XML or other proprietary formats. Regardless of their representation method, models can be used to generate software, to automate design, refactoring, deployment, builds, debugging and testing [1].

Making use of models to raise the level of abstraction and to automate software development has been a long-term vision of last decade. To realize this vision general purpose modeling languages such as UML has become industrial standards. However, UML is primarily designed for documentation, not automating development, thus one can claim that UML has not provided many improvements to raise the level of abstraction due to its lack of precision, numerous modeling concepts, poorly defined semantics and weak extensibility mechanism [1, 52]. As a result, it is difficult to learn and apply UML in an MDD environment [52]. Commercial MDD tools would be based on technologies designed for development, not documentation [1]. Any artifact that contributes to actual software development must be capable of manipulation digitally. Thus, a model must have a precise syntax, a comprehensible semantics, and well-defined mappings to source code or other models. Furthermore, the UML 2.0 specification still does not address critical development issues such as database design, testing, deployment, component-based development, and user interface construction [3]. Moreover, models must be validated so that faults can be discovered before developers transform the models into implementations. Existing formal verification techniques can be applied, but few of them can handle large system models [52].

2.3 Benefits of Model-Driven Engineering

In this section, the author has discussed the benefits of MDE considering the attributes such as productivity, scalability and efficiency. MDD combines the scalable aspects of agile approaches with other techniques to prevent quality and maintainability problems in large-scale system [68].Potential benefits of MDD are based on following principles [68]:

- Development of DSLs that are oriented more towards the problem space than the solution space facilitates formalization and condensation of software designs.
- Abstraction from the level of expressions of today's programming languages and platforms.
- Use of generators to automate routine software development activities.
- Separation of concerns which allows separate processing and evaluation of functional and technical code.
- Reusability across project boundaries via the formation of software product lines.

Völter et al [68] summarize the economical benefits of above principles as follows:

- Software product lines shorten the development life cycle.
- Separation of concerns makes technology evolution easier. Thus, new implementation technologies can be adapted via automation at lower cost. Technology changes do not affect the application models.
- Shorter time-to-market and less human resources.
- Lower entire product life cycle and maintenance costs, higher customer satisfaction.
- New business requirements are implemented faster by employing domain knowledge rather than technical knowledge.

Selic [56] compares the efficiency of automatically generated code to the efficiency of handcrafted code. Code efficiency can be decomposed into two separate areas: performance (throughput) and memory utilization. Current model-to-code generation technologies can generate code with both performance and memory efficiency on average, within 5 to 15 percent (better or worse) of equivalent handcrafted systems. As the technology evolves and MDE tools become more mature, the situation can improve [56].

Since MDE is intended for large-scale industrial applications, the scalability of current tools and methods is a critical issue. The important metrics of concern are *compilation time* and *system size*. Compilation time can be divided into two separate parts: *full-system generation time* and the *turnaround time* for small incremental changes. Since small changes are much more frequent during development, turnaround time has greater impact on productivity in such a way that if a small change requires regenerating a large part of the code; it can slow down development to an unacceptable level. To deal with this, code generators should have change impact analysis capabilities that minimize the amount of code generation [56].

The afforded benefits of MDE in an industrial case study are summarized as follows [57]:

- Design models are easier and faster to produce and test.
- Labor-intensive and error-prone development activities are automated.
- Design effort is focused on applications rather than on platform details.
- Reuse of design and test artifacts between platforms or releases is enabled.
- Design models are more stable, complete and testable.
- Design models can be verified through simulation and testing.
- The learning curve for new staff is shortened.

This industrial application of MDE [57] shows that generated code requires fewer inspections than handwritten code and efficiency of inspections increases considerably. Furthermore, code automation on average results in a five-fold increase in terms of number of source lines produced per staff months throughout the development life cycle. On the other hand, the increase in design effort is compensated by the significant reduction in coding effort. When the code generators become more mature, number of errors introduced in generated code decreases significantly. Automation of labor-intensive and error-prone development activities results in additional productivity improvements such as a significant reduction in the turnaround time for defect fixes during the test execution process. Incorporating platform specific aspects into the code generator allows engineers to experiment alternative implementation strategies, software architectures, or hardware choices at lower costs [57].

Separation between domain-specific information and application information contributes to efficiency of team structures in such a way that domain experts capture their knowledge in code generators, and the application development teams can focus explicitly on design and testing issues. Verification of design models using simulation and other techniques results in significant quality improvements. For instance, requirements defects can be discovered early in the development life cycle and in this way the cost to fix these defects decreases significantly [57].

Attributes	**Metrics**	**Benefits**
Efficiency	Performance, memory utilization.	5 to 15 percent increase of equivalent handcrafted systems.
Scalability	Compilation time, system size.	Shorter full-system generation time and turnaround time.
Productivity	Size/Effort, number of change requests.	A five-fold increase on average in terms of number of lines of code per staff months, less number of change requests.
Reliability	Number of defects introduced in a period of time.	Less defects in generated code.

Table 2 Quantitative benefits of MDD [56, 57]

As it is summarized in Table 2, the quantitative benefits of MDD can be measured considering attributes like efficiency, scalability and productivity and their corresponding

metrics. In addition to quantitative benefits, MDD also provides qualitative benefits such as increase in staff morale due to automation of menial tasks, which will lead to quantitative benefits in long-term. The author believes that organizations should realize that MDD is a long-term investment and actual business benefits can only be revealed after completion of long-term projects that employ appropriate MDD tools.

Besides its potential significant benefits, the main drawback of MDD is the lack of sufficient tool support [78]. Since current MDD tools are still under development and there is a need for more stable releases, potential built-in faults in these tools can pose a threat against the validity of defined models and the reliability of the generated source code. Another drawback is the consideration of security requirements in the development process due to the difficulty to analyze and model security requirements and to integrate them with general software engineering models [78].

2.4 Software Factories

Greenfield and Short [1] defines a Software Factory as a configuration of languages, patterns, frameworks, and tools that can be used to rapidly and cost effectively produce an open-ended set of unique variants of an archetypical product. A more detailed definition is as follows [1]:

*"A **software factory** is a **software product line** that configures extensible tools, processes, and content using a software factory template based on a **software factory schema** to automate the development and maintenance of variants of an archetypical product by adapting, assembling, and configuring framework-based components."*

A software factory is consisting of a software factory schema and a software factory template based on the software factory schema. A software factory schema is a description of a software factory which defines the relationships between development artifacts such as models, configuration files, source code files, test case definitions and deployment manifests. The software factory template configures extensible tools, processes and content to form a product facility for the product family. A software factory template includes code and metadata that can be loaded into extensible tools such as an Integrated Development Environment (IDE) to automate development and maintenance of family members. While configuring a schema customizes the description of the software factory, configuring a template customizes the tools and development environment used to build the family members [1].

2.4.1 Building a Software Factory

The goal of building a software factory is to rapidly develop members of its product family. Since a software factory is a kind of software product line, building a software factory shown in Figure 2 includes the following activities [1]:

- Building a *software factory schema* including product line analysis and design. A software factory schema is a collection of viewpoints (such as Web service, Web page layout, and business entity viewpoints) and each viewpoint describes a specific aspect. Product line analysis includes definition of a scope that identifies the software products that the software factory will develop. Product line analysis produces product line requirements organized into viewpoints. A viewpoint defines a perspective from which a given aspect of a software product can be described, and provides a pattern for producing such a description. Product line design defines how the software factory will develop products within its scope. In product line design, architecture for the target software family is designed. It supports variation among the family members. Product line design produces the following artifacts:

requirements mapping, product development process and a plan for using tools to automate some parts of the product development process [1]. In this sense, a schema describes the product line architecture and key relationships between components and frameworks of which it is composed. A schema contains information about source code directories, SQL files, configuration files, what DSLs should be developed, how models can be transformed into source code and other artifacts [8].

- Building a *software factory template* that instantiates the software product line. It is a specialization of product line implementation. During asset packaging activity the patterns, tools, processes and other production assets are packaged as a software factory template [1]. In this sense, a software factory template provides patterns, guidance, frameworks, samples, and custom tools such as DSL visual editing tools, scripts and other ingredients used to build the product [8].
- Once the software factory schema and template is built, *product development* task can be performed by product developers. Product development is organized by the development process that was defined by product line design and later on customized during the configuration of the software factory schema for the product under development. During product specification, product requirements are specified in terms of product line requirements and mapped to the other production assets [1]. The software factory schema, the variable and fixed assets such as patterns, frameworks, tools and processes that constitute the software factory template are loaded into an extensible development environment such as Visual Studio Team System [1]. Development environment can be configured with the software factory template, and later it becomes a software factory for a given product family [8].

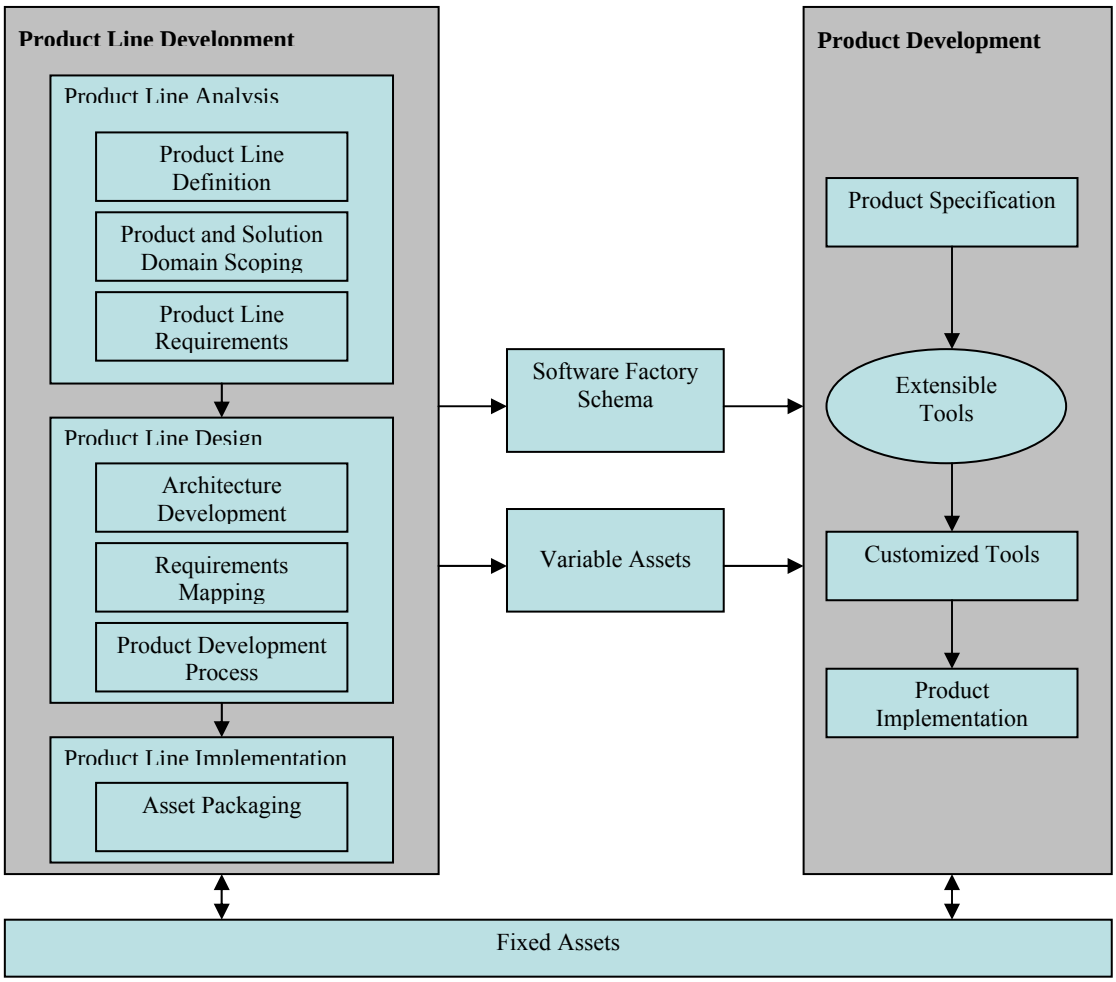

Figure 2 Building a Software Factory [1]

Software factories move software development toward industrialization. Software factories have many implications on software industry. It provides productivity gains not only in broad horizontal domains such as user interface construction and data access, but also in narrow vertical domains such as banking and health care. One of its implications is in the development of domain-specific assets. Product developers will rarely use general-purpose programming languages; instead, they will use customized tools to assemble framework based components using domain-specific languages. These tools and languages will be built cost-effectively based on domain expertise. Software factories paradigm does not propose to commoditize software development that it will no longer be a creative process; to the contrary it proposes automation of routine and menial tasks and freeing you up to focus on creative tasks [1].

Software factories integrate critical innovations to define a highly automated approach to software development. These critical innovations include [1]:

- Building families of similar but distinct software products to enable a more systematic approach to reuse.
- Assembling self-describing service components using new encapsulation, packaging and orchestration technologies.
- Developing Domain-Specific Languages and tools using code generation that reduce the amount of handwritten code.

2.5 Model Driven Architecture

The latest definition of MDA was approved by [11]:

"MDA is an OMG initiative that proposes to define a set of non-proprietary standards that will specify interoperable technologies with which to realize model-driven development with automated transformations. Not all of these technologies will directly concern the transformations involved in MDA. MDA does not necessarily rely on the UML, but, as a specialized kind of MDD (Model Driven Development), MDA necessarily involves the use of model(s) in development, which entails that at least one modeling language must be used. Any modeling language used in MDA must be described in terms of the MOF language, to enable the metadata to be understood in a standard manner, which is a precondition for any ability to perform automated transformations."

MDA is built around UML and MOF and targets reuse of existing UML tools by means of UML profiles. UML profiles are used in cases when UML is not sufficient to represent a given problem domain. MDA recognizes three types of models: Platform Independent Model (PIM), Platform Specific Model (PSM) and Platform Description Model (PDM). A PSM is generated from a PIM by an automatic transformation. A platform is considered as a particular implementation technology. For each platform there is a direct mapping from the PSM to the given platform's source code [8]. However, Cook [29] argues that OMG's definition of "platform" is vague, and the only real example is J2EE. There are few standardized mappings between platform-independent and platform-specific models and they only targets J2EE platform [29].
MOF is domain-specific modeling language designed for defining modeling languages. Thus, a MOF model is a definition of an MDA modeling language. It is also a mechanism that determines how a model defined in an MDA modeling language can be serialized into an XML document. A modeling language for a particular domain should address many aspects. It must define domain concepts and represent them either diagrammatically or in text. In addition, it must define how the user may interact with the language and how the models will be validated and interchanged. However, MOF only defines the basic concepts in a language and how models of those concepts can be stored and interchanged. A MOF description of a language tells little about how the user interacts with the language. MOF is

primarily a technology for the storage of conceptual models and interchanging them using XML Metadata Interchange (XMI) [14], another OMG standard [29]. XMI defines production rules for generating XML schemas from MOF metamodels [1]. If the concepts of a language are defined using MOF, XMI can be used to generate XML-based serialization for the language [29]. XMI is strongly coupled to the metamodel of the target language, thus the interchange format cannot be changed unless the metamodel is changed [1]. Cook [29] and Greenfield [1] argues that XMI claims to provide interoperability of modeling tools, however interchange of models is problematical and XMI documents are verbose and difficult to read. Thus, Cook [29] cites that the right approach for serializing a particular modeling language is to purpose-build a schema for that language, and to provide tools that explicitly manage and automatically interpret the mapping between the language and its serialization format. In conclusion, using MOF standard as the language for designing MDD tools would lead to practical consequences on those tools and the missing elements of the MOF would continue to introduce major changes in its specification. In addition, model serialization approach of MOF does not fulfill its goals [29].

2.6 Comparison of Software Factories and Model Driven Architecture

In this section, the author has highlighted the important differences between two MDD approaches. MDA tends to focus on UML-based modeling languages; however MDD does not have these restrictions. The main goal of MDA is interoperability between tools and the long-term standardization of models for popular application domains [68]. In this sense, the main difference between software factories and MDA is that Software Factories are less concerned with portability and platform independence, and more concerned with productivity [1, 8]. Software Factories and MDA are compatible approaches, but neither OMG nor Microsoft recommends integration of two approaches. The selection depends on the circumstances and final success of application of a particular approach can depend on marketing/commercial reasons [32].

Greenfield [1] argues that a number of issues that are addressed in Software Factories are not concretely addressed in MDA. These are: How software is developed using models, how to define relationships between models, how to define graphs of interrelated models that help to solve specific problems, how the models interface with code, how are models versioned and partitioned, how models relate to architectural styles, architectural description standards, patterns, or frameworks, how models and modeling tools can be reused systematically, how models fit into the rest of the development cycle and how the cost of models can be reduced by making it cheaper and easier to define them. MDA does recognize the need for domain-specific languages, but it does not identify the need to focus on software product families, because it mainly focuses on one-off development. In contrast, Software Factories approach defines a family-based development artifact called a software factory schema that defines viewpoints from which the family members are modeled. It also defines the DSLs, patterns, frameworks, tools and other assets used to capture metadata from each viewpoint, and the mappings between models required to support transformation and other forms of automation. Comparing to MDA, Software Factories provides much better traceability through the software factory schema, in this way affects of changes in domain can be assessed more rapidly and reliably [1].

Greenfield [1] cites that mature aspects of Software Factories approach are language technology, tool extensibility, pattern composition, deferred encapsulation, and development of standard DSLs, patterns, frameworks, and tools for popular domains. On the other hand, Greenfield [1] argues that despite increasing integration of modeling tools with IDEs, IDEs are not yet mature in terms of providing automation using popular modeling technologies. Most IDEs provide simple code visualization and low-level code generation from general-

purpose models such as UML class models. Some IDEs are starting to include metamodeling facilities that can be used to develop DSLs. These IDEs should provide robust extensibility mechanism that can be used to develop editors, compilers, debuggers, and other tools that make DSLs available to product developers [1].

Demir [27] performs a comparison of MDA and Software Factories shown in Table 3 focusing on the development activities, advantages, disadvantages and applicability of both. In order to apply two approaches, a web-based application is developed first applying MDA and then Software Factories. The experiences gained during the development phases are compared, analyzed and evaluated. Demir [27] argues that depending on their intended purpose, both approaches demonstrate strengths and weaknesses. In order to guarantee each approach's benefits, an appropriate domain, professional developers, a suitable tool and a precise definition of the intended products are needed. Demir [27] highlights the following differences:

- Depending on the utilized tool's role, MDA provides productivity increase due to the omission of the implementation phase. On the other hand, learning phase for the MDA tool is time-consuming due to its complexity [27].
- Due to same reason as MDA, Software Factories approach can increase productivity as well. However, the expense for the Software Factories approach is much higher than MDA due to necessity of development of DSL which requires a lot of time and expert knowledge about the problem and solution domain [27].
- DSL can provide better efficiency, because it comprises domain concepts and thus it is closer to the problem domain. Furthermore, DSL development facilitates involving business stakeholders into the specification process to avoid ambiguities and misconceptions. This is an advantage for DSLs, since UML experts are required in development of UML models [27].
- Both approaches provide improvement in quality and reliability of the system due to the reason that generated code is less error-prone and the generated applications exactly meet their specification in form of models (code generation is performed according to a scheme, rules or code-templates) [27].
- Due to the cost of developing a DSL and a code generator, Software Factories approach can only be recommended for Product Line Development. Once a DSL and code generator is developed, the costs for developing similar, but distinct product family members are very low. The overall costs of a product line development can be distributed among all members; this makes Software Factories approach productive [27].
- On the other hand, for one-off development, the SF approach would be expensive due to required time, budget and resources. MDA approach can be used for both one-off development and product line development. In Product Line Development, MDA approach provides productivity increase in such a way that first models can be reused for other product family members [27].
- While MDA focuses on platform independence, Software Factories is an entire software development paradigm and focuses on product line development. MDA is based on UML which has to be specialized with UML Profiles to be appropriate for MDD. To the contrary, a DSL is developed for a specific domain from the beginning without specialization. In this sense, DSLs can be very efficient for a particular domain, but also very useless for other domains [27].

	MDA	**Software Factories**
Initial cost	Low	High
Productivity increase	Low	High
Efficiency increase	Low	High
Reliability increase	High	High
Applicability	Product Line Development One-off development	Product Line Development

Table 3 A comparison performed by Demir [27]

3 DOMAIN-SPECIFIC MODELING

The goal of this chapter is to provide current state-of-art in Domain-Specific Modeling. It is structured as follows: Section 3.1 presents the concepts behind Domain-Specific Modeling and how to build support for it by introducing different phases and roles. Section 3.2 presents Domain-Specific Language development, its related benefits and risks and performs a comparison of Domain-Specific Languages and UML which is a General-Purpose Language. Section 3.3 presents two major MDE platforms, Microsoft DSL Tools and Eclipse Modeling Project, how they provide support for Domain-Specific Modeling and how models defined using two modeling technologies can be exchanged.

3.1 Introduction

An independent organization DSM Forum [35] exists to spread the knowledge and know-how of Domain-Specific Modeling (DSM). DSM Forum embodies its vision in the following statements [35]:

"Domain-Specific Modeling raises the level of abstraction beyond programming by specifying the solution directly using domain concepts. The final products are generated from these high-level specifications. This automation is possible because both the language and generators need fit the requirements of only one company and domain. Industrial experiences of DSM consistently show it to be 5-10 times more productive than current software development practices, including current UML-based implementations of MDA. DSM does to code what compilers did to assembly language." Besides this vision, more investigation is needed in order to advance the acceptance and viability of DSM.

Selection of a domain is first step towards development of domain-specific languages. Selecting a domain implies tradeoffs between more general applicability of the DSL and more specificity [60]. In another words, a tradeoff between the focus and size of the language is needed. A language that represents a larger domain can weakly specialize to any particular aspect of the domain. To the contrary, a language that represents a small domain may have a limited number of target users [62]. Völter [60] distinguishes two kinds of software domains:

- Technical (or Horizontal) domains address key technical issues related to software development such as Distribution, Persistence and Transactions, Graphical User Interface (GUI) Design or Concurrency.
- Functional (or Vertical) domains address business issues such as Banking, Human Resource Management and Insurance.

A typical software system consists of several domains mentioned above.

Kelly [28] argues that attempts to make a completely generic modeling language and generators have failed due to the reason that raising the level of abstraction means sacrificing fine control and complete generality. In this sense, if the modeling language is too broad, it is likely that communities would use disjoint subsets of the language with no real communication between them. When a modeling language becomes larger and more complicated, it becomes more difficult to understand models defined in that language [62]. Thus, the only way to generate full code directly from models is to make both the modeling language and generators domain-specific [28].

3.1.1 Building support for Domain-Specific Modeling

Building support for DSM with full automatic code generation includes three phases [35]:

- *Assembling the domain-specific component library*: A domain-specific component library is not always necessary, but it facilitates code generation development significantly. Often, code components already exists from previous development cycles, thus they can be reused. Components can be either developed in-house or acquired as third-party components [35].
- *Developing the domain-specific modeling language and editor*: In this phase, domain experts combine knowledge and experiences from multiple resources such as hints from component library, domain rules and architects. Domain expert should also consider how to achieve a good mapping from the domain metamodel to combinations of components for the sake of code generation. Metamodeling languages can be applied to describe both domain rules and their mappings. In the development of a DSM language, tool support plays a vital role. Tool support allows the domain expert to concentrate more on the difficult task of developing the method and the modeling tool implementation of it [35]. On the other hand, building a CASE tool from scratch for a simple modeling language would take man-years, thus existing tools for building DSM editors should be considered first [28].
- *Developing the code generator*: In this phase, the domain expert specifies how code using the components can be automatically generated from the structures in the users' models. The DSM tool should provide the necessary functionality for creating such generation scripts, and should guide the expert by allowing him to reference and use the concepts in the metamodel. Furthermore, the end user should be able to use the code generation simply, in this way the cost of developing the code generator can be defrayed over many users [35].

Krahn et al [47] separate activities mentioned above and assign them to specific roles within an agile development process (see Figure 3):

- A *language developer* defines or enhances a DSL in accordance with the needs of product developers. The language developer not only defines the syntax of modeling language, but also describes its meaning in terms of semantics and ensures that newly added concepts are properly integrated in the existing language and provides a manual for their use [47].
- A *tool developer* develops code generators for the DSL which includes the generation of production and test code as well as the analysis of the content and its quality. Furthermore, tool developers integrate newly developed or reused language processing components and generators to form tools used with in project. If a commercial off-the-shelf tool is used, language and tool developer roles are unnecessary [47].
- A *library developer* develops software components or libraries, in this way simplifies the code generator. This role is closely related to the tool developer but requires more domain knowledge. Component libraries encapsulate detailed domain knowledge and provide a simplified interface for the needs of code generation [47].
- The *product developers* employ the provided tools for different activities within the project. They specify a solution using their domain knowledge expressed in DSLs [47].

In the case of MDA, an additional role, named *transformation definition engineer* is needed due to the transformational nature of this approach [47].

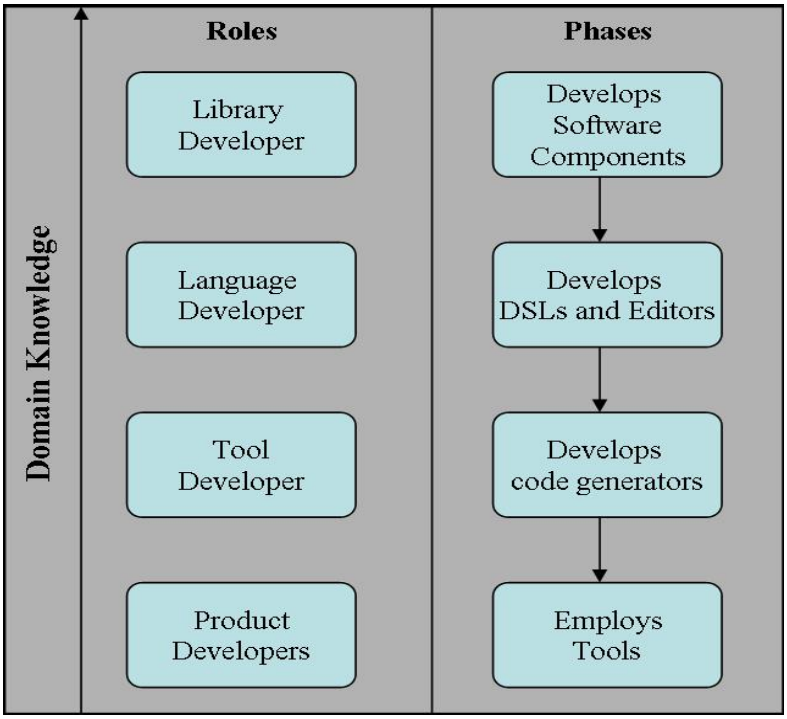

Figure 3 Phases and associated roles in building support for DSM [35]

3.2 Domain-Specific Languages

General-purpose language (GPL) refers to a language that encodes generic abstractions. To the contrary, domain specific language refers to a language that encodes abstractions used in a narrow domain [4]. A domain specific modeling language raises the level of abstraction and bridges the implementation closer to the vocabulary understood by domain experts, engineers and end-users. A DSL enables the specification of software from a specific viewpoint. A GPL such as UML is not viewpoint-based; instead it supports only generic, undifferentiated specification [1]. Even though UML 2.0 allows users model system from four viewpoints (static structural, interaction, activity and state), it currently does not provide mechanism for creating models using user-defined viewpoints which help developers to model complex system [52]. Today business applications cover multiple functional domains, thus multiple DSLs are needed to describe them [1].
Examples of popular DSLs include:

- SQL: A structured query language provides an interface to relational database management systems.
- HTML, a markup language for the creation of web pages.

DSLs and Domain-Specific Modeling Languages (DSMLs) are considered to be the same and the terms are usually used interchangeably. DSL is represented by a metamodel and its models conform to that metamodel. DSLs could be the right technique to develop software in shorter time with better quality. DSLs are considered as a good solution to the problem of reuse both on technical and on architectural and design level [49].

3.2.1 Basic criterion for DSLs and DSL tools

Even though there is a lack of general theory for designing modeling languages and corresponding tools [51], a DSL should fulfill all these basic criteria [1]:

- The concepts represented by modeling language must be understood by the experts familiar with its purpose. For instance, a modeling language for developing web-services should contain concepts such as web methods and protocols. Another example could be a

modeling language used to visualize and edit C# source code should contain concepts such as classes, methods, properties, delegates and events.

- Graphical or textual notation of the language should be easy to use. For example, it should be easy to represent an inheritance hierarchy in a language used to visualize and edit C# source code.
- The modeling language should have a well-defined set of rules or grammar which can be used by tools to validate expressions.

Based on the criteria mentioned above, DSL tools should include [1]:

- Graphical editors for creating and maintaining specifications.
- Language processors, compilers or interactive generation environment for producing implementation from models.
- Tools that automate other development tasks using metadata captured by models based on the language. These tasks include test generations, configuration management, measurement and deployment.

Precisely defined DSLs and DSL-based tools can be used for [3]:

- Precise abstractions from which code is generated.
- Precise abstractions that map to variation points in frameworks and components.
- Precise mappings between DSLs.
- Conceptual drawings that have precisely specifiable mappings to other DSLs or to code artifacts.

3.2.2 How to make a DSL usable

Feilkas [49] cites the tasks that have to be carried out to make a DSL usable:

1. *Definition of an abstract syntax*: Many DSL-tools allow the definition of the abstract syntax as a metamodel. This metamodel is defined by a data modeling technique similar to class diagrams or Entity Relationship (ER) diagrams [49].
2. *Definition of a concrete syntax*: For every language element there has to be a graphical symbol that represents the abstract model element. In the case of textual languages, both abstract and concrete syntax can be described by a grammar which specifies terminals, non-terminals and production rules [49].
3. *Definition of semantics*: The semantics of the DSL is defined by implementing the generator backend and giving translational semantics into some target language which already has some behaviour definition for its elements [49].

There are three kinds of approaches to realize the generator backend [49]:

- *Templates* are the most preferred approach to code generation in DSL-tools (language workbenches). In this approach, code-files in the target language are considered as basis. Expressions of a macro language that specify the generator instructions are inserted. Ordinary programming languages are used to specify the behaviour of the generator (C# in the MS/DSL Tools) [49].
- *Patterns* approach allows specifying a search pattern on the model graph, in this way for every match a specific output is generated [49].
- *Graph-traversing* approach specifies a predetermined iteration path over the syntax graph. For every node type, generation instructions are defined which are executed every time such a node is passed. This approach is used in classical compiler construction and textual languages [49].

Feilkas [49] cites that the generation techniques that are used by DSL-tools do not respect target language syntax. In this sense, a formal mapping between the source and target language elements is crucial to specify the semantics of a newly developed DSL. By specifying a translation to a target language that has operational semantics a newly developed DSL implicitly gets a formal definition of its own semantics [49].

3.2.3 DSLs as Software Products

DSLs can be considered as software products in a sense that a family of DSLs can be defined by developing common features with well-defined variation points. A tool factory shown in Figure 4 is a software factory used to build tools that manipulate and process the members of a family of DSLs from specifications of the family members. The design of language X is designed in the Language Designer from the language fragment, Y, and language design pattern, Z. Design Tool Generator generates part of a design tool for language X. Tools are available that implements some of this structure. These tools are often called “meta-tools”, because they interpret a metamodel to implement a language. There are some pitfalls that come with in commercial meta-tools: Since meta-tools usually interpret language definitions, instead of compiling them, it often leads to performance, scalability and error-handling problems in large models. Instead of interpreting language definitions, a better approach is to provide a library of language definition components in an IDE and to allow the user assemble languages from them [1].

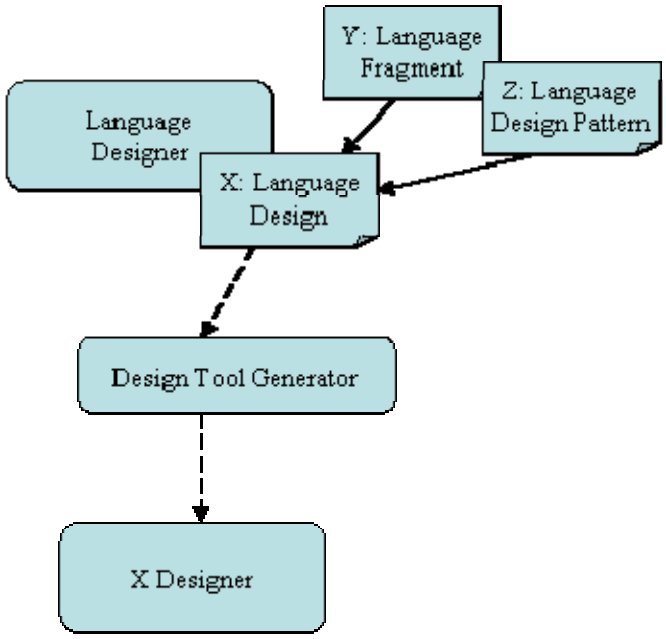

Figure 4 Overview architecture of a tool factory [1]

3.2.4 Benefits and Risks of DSLs

Good DSLs are often small and simple refer to the term “little languages”. To make it useful, a DSL should be designed as a limited language tightly focused on a single problem [31]. DSL development involves both risks and opportunities. The main benefits include:

- Since DSLs allow solutions to be expressed at the domain level, domain experts can understand, validate, modify and even develop domain models expressed in DSLs [38].
- DSLs are concise, self-documenting to a large extent, and can be reused for different purposes [38].
- DSLs are small languages and very focused on a specific tasks and have very well defined semantics, thus they can be easily tooled [1].
- DSLs enable developers to separate previously connected development activities for a software system. In this way, they can concentrate on a single task at a time which leads to better results and more efficient development process [4].

- DSLs enhance productivity [67], maintainability [1], reliability [67], portability [38] and reusability of software artifacts [3].
- DSLs enable expression, validation and optimization of concepts at the level of abstraction of the problem domain [3, 38].
- A DSL can facilitate the construction of complex test cases, thus it improves testability of software [38, 39].

On the other hand, potential risks include:
- The high expense of designing, implementing and maintaining DSLs and tools [1].
- The high expense of training DSL users and migrating developer skills [1, 38].
- The difficulty of finding the proper scope for a DSL [38]. DSLs are limited both in scope (they refer to a particular domain) and capability (they lack features that are basic for general-purpose languages) [31].
- The difficulty of balancing between domain-specificity and general-purpose programming language constructs [38].
- The potential loss of efficiency comparing to the manually written source code [38, 67].

DSLs can be implemented as standalone languages or extensions to existing languages [4]. Since it is costly to develop new DSLs from the scratch, instead existing languages like XML can be extended. Instead building new compilers, editors, and debuggers from the scratch, plug-ins for existing IDEs such as .NET or J2EE can be built [1].

In order to mitigate the risks involved in building DSLs, organizations can leverage the skills of their most talented developers by making them product line developers, who build the tools and languages used by product developers. In this way, organizations will have better technology for harvesting, refining, documenting, testing, packaging, and supporting reusable assets [1].

To elaborate discussion, Fowler [36] divides them into two broader styles: internal (e.g. Lisp) and external DSLs (e.g. XML configuration files). External DSLs are written in a different language than the main (host) language of the application and transformed into it using some form of compiler or interpreter. The advantage of an external DSL is its ability to express domain in the easiest form to read and modify. In addition, an external DSL can be evaluated at runtime; in this way commonly changed parameters can be altered without recompiling the program. Then, a translator should be built to parse the configuration file and produce something executable. However, when the language gets more complex, the cost of building translator increases. The major disadvantage of external DSLs is that they lack *symbolic integration* meaning that the DSL is not linked to the base language [36]. On the other hand, internal DSLs morph the host language into a DSL itself. In contrast to external DSLs, internal DSLs eliminate the symbolic barrier with the base language and make the full power (tooling) of base language available. The disadvantage of internal DSLs is that they are limited by the syntax and structure of the base language [36].

3.2.5 Trade-offs of using DSLs

Fowler [36] cites that the critical issue in language oriented programming is to make trade-off between the benefit of using DSLs and the cost of building the necessary tools to support them effectively. Using internal DSLs reduces the cost of developing tools, but resulting constraints on the DSL itself can also reduce the benefits. On the other hand, an external DSL brings the high potential to realize its benefits, but it also comes with high costs to design the language, build the translator, and tool support [36]. This lead to the creation of new category of tools, named language workbenches [36] which use IDE tooling to make language-oriented programming a viable approach [38].

Feilkas [49] argues that today many languages are developed that are not fully compatible, thus manual coding is still needed to get an executable program. To get highest benefit from DSLs, the target code does not need to be touched after generation. Otherwise, the product developers using the DSL must still have full knowledge of the platform and the architecture of the generated code. The reason for manual coding in generated code is the difficulty to specify languages that are capable of expressing more than architectural and design decisions such us component and service structures. On the other hand, specifying logical or arithmetical expressions in a DSL leads to difficulties in its metamodel specification. Besides, manual modifications in generated code may destroy architectural decisions specified in the DSL and its generator. In addition, when the generator needs to be rerun due to the changes in the model, manually written code might be lost. These kinds of problems become obsolete if full code generation could be achieved from the DSL specifications [49].

3.2.6 Comparison of UML and DSLs

Nowadays two approaches appear on the top of MDE: UML [13] and DSLs. UML provides a rich set of modeling notations and semantics and allows developers to customize it to their specific domain by using extension mechanism such as stereotypes, tagged definitions and constraints. A consistent set of such extensions is called an UML profile. UML profiles are used to model some aspects of the system that can not be directly represented by native UML elements. Another approach to represent applications is to specify each aspect separately; this approach is called Domain Specific Modeling (DSM). A DSL provides appropriate notations and abstractions to describe a specific domain. Its semantics are precise and ambiguous. Both approaches have advantages and drawbacks which are discussed further in this section [12].

Greenfield [1] argues that UML as a general-purpose modeling language cannot possibly address the requirements of all possible domains in a manner that provides real meaning. Furthermore, UML does not clearly define the requirements it is intended to address due to its wide scope. The UML standard does not appear to have been designed to support tools, thus when implementing UML compliant tools, deviation from the standard is a necessity [1].
The applicability of UML models to model-driven development is widely argued. General programming languages like UML have not increased productivity, since the core models are on the same level of abstraction as the programming languages supported. UML tries to be everything to everybody, thus it cannot raise level of abstraction above the lowest denominator. UML-based MDA tools have not increased productivity significantly either [35]. An independent case-study [43] of UML-based MDA showed productivity increases of 35%, it is far from 5-10 times more productivity gained from DSM [28]. To increase productivity, the base UML can be extended with domain-specific enhancements, or replaced with new MOF-based metamodels. However, UML tools are not designed to support extensibility.

Cook [29] argues that if the UML model is to be used as first-class development artifact, the mismatches of UML models to the implementation technologies must be resolved. For instance, a UML class cannot be used directly to depict a C# class because a UML class does not support the notation of C# class properties. In the same manner, a UML interface cannot be used directly to depict a Java interface because a UML interface does not contain static fields of a Java interface [29]. To overcome such mismatches, Cook [29] suggests to either violating UML standards or introducing uncomfortable transitions into the development process. UML standard should be formulated as a set of flexible and extensible diagramming conventions which allow the appropriate technology mappings to be developed seamlessly without the mismatches [29]. Cook [29] also argues the extensibility mechanism

provided by UML Profiles does not permit a seamless mapping from UML into target technologies.

In a UML Profile, model elements are extended using stereotypes that define additional element properties [52]. Profile examples are OMG's UML Profile for Schedulability, Performance and Time and UML Profile for QoS (Quality of Service) and Fault Tolerance [53]. UML 2.0 profile mechanism does not provide means for precisely defining semantics that are associated with extensions. Thus, developers cannot use profiles in their current form to develop domain-specific UML variants that support the formal model manipulations required in an MDD environment. Since UML does not provide the base elements for DSM, its extension mechanisms can be used to change the UML metamodel. These heavyweight mechanisms employ MOF to define the metamodel for the UML variant.

In order to address interoperability issue between DSM and UML-Profiling approaches, Abouzahra et al [12] proposes a tool which acts like a bridge between UML profiles and DSLs. To implement such a tool, they used the functionalities provided by AMMA (ATLAS Model Management Architecture) which is a new generation MDE platform built on the top of EMF. The implemented tool takes as input a UML profile and the metamodel of the system described by the profile. It transforms UML models designed in this profile to models conforming to the profiled metamodel. The tool aims to generate transformations between models automatically [12].

On the other hand, the UML 2.0 metamodel can make understanding, using, extending and evolving the metamodel labor-intensive task. Thus, the task of defining model transformations using the UML metamodel is tedious and error-prone. To overcome this problem, simplified metamodel views for each UML 2.0 model type can be used. These views should define only the concepts and relationships that appear in the models. Tools that query the metamodel and extract specified views from it can be used to better understand the UML metamodel and to obtain views for specifying patterns and transformations [52].

In conclusion, UML 2.0's size and complexity brings a hurdle not only to UML-based MDD tool developers but also to OMG working groups involved in evolving the standard. The metamodel needs to be reconstructed to facilitate evolution and tools should be developed to help navigate the metamodel [52].

3.3 Domain-Specific Modeling Tools

In this section, the author has presented two major industrial Model-Driven Engineering tools with respect to level of support they provide for DSM. Pelechano et al [32] cites the basic facilities that a Model-Driven Development tool must provide:

- Metamodeling Facilities
- Model Persistency
- Graphical Notation Development Tools
- Model to Text and Model to Model Transformation Tools

In the next sections, Microsoft DSL Tools and Eclipse Modeling Project have been investigated to determine how much support they provide for above facilities.

3.3.1 Microsoft DSL Tools

Visual Studio 2005 Team System modeling vision is to change the way the developers perceive the value of modeling. Models should not be just a piece of documentation waiting to become outdated. When models are considered as first-class development artifacts, developers write less conventional source code due to high level abstractions that are employed. In addition, others involved in development such as business analysts, architects and designers will perceive modeling as adding value to their tasks. MS/DSL Tools can be

considered as “tools to build tools”, facilitates the task of defining DSLs and reduces the effort and skills needed to build graphical editors and compilers for them [3].

“The Microsoft Tools for Domain-Specific Languages is a suite of tools for creating, editing, visualizing, and using domain-specific data for automating the enterprise software development process” [26]. MS/DSL Tools employ the Software Factories approach that is introduced by [1] and extend on the Microsoft’s notation for Domain Specific Modeling which is a specific realization of MDE standards and principles [5]. When this thesis was written, the latest release of MS/DSL Tools was September 2006 CTP for Visual Studio 2005 SDK version 3.0 [19].

The tool suite consists of [26, 30]:

- **A project wizard** for creating a fully configured solution. In this solution, one can define a domain model that consists of a designer and a text output generator. Running a completed solution within Visual Studio opens a test solution in a separate instance of Visual Studio. In this way, the designer and the text output generator can be tested.
- **A graphical designer** for defining and editing domain models. A screenshot of this designer is shown in Figure 5.

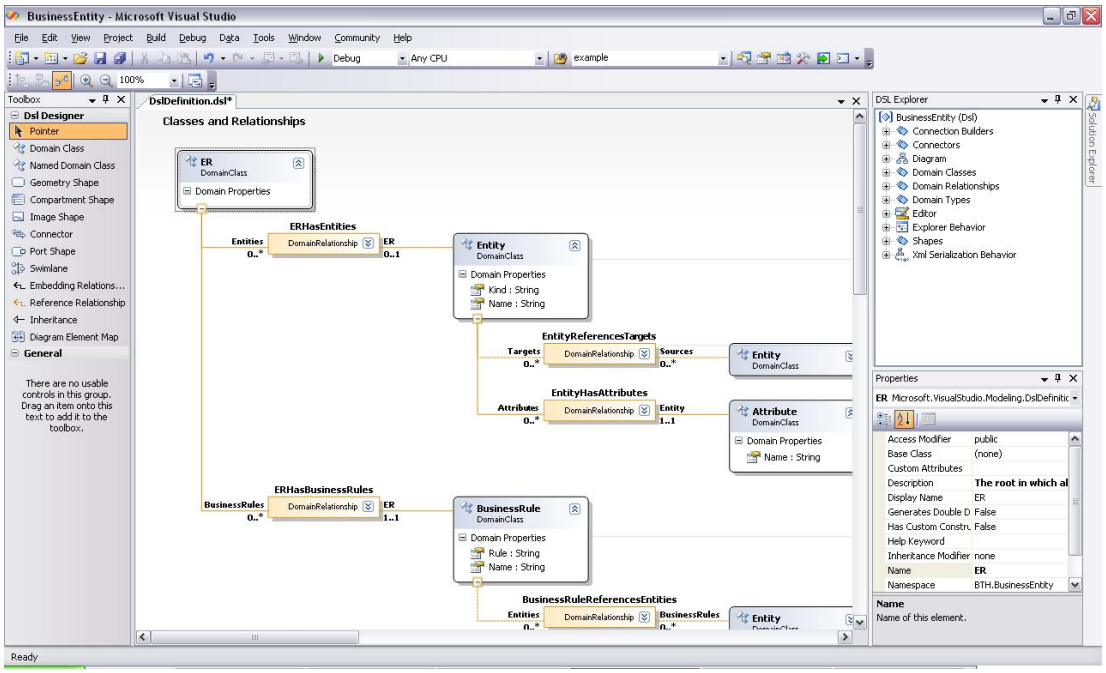

Figure 5 Domain Model Editor provided by MS/DSL Tools

- **Designer definitions in XML.** The source code for implementing designers is generated from these definitions (without any manual programming). In this way, a graphical designer hosted in Visual Studio can be designed without any hand coding.
- **A set of code generators**, which take a domain model definition and a designer definition as input and produce source code that implements both components as output. In addition, code generators validate the domain model and the designer definition, and they rise errors and warnings accordingly.
- **A framework for defining template-based text output generators**. These generators take data (models) that conform to a domain model as input and generate text output that is based on the template. Parameters in the template are substituted using the results of running a C# script that is embedded in the template. Supported target languages for these templates are C# and Visual Basic.

Microsoft uses the term “domain model” as equivalent of a metamodel. It is composed of a class hierarchy and relationships. A relationship may be a simple reference or an embedded relation (composition in UML terms). It has properties, two roles (source and target) and may have a supertype. A model is an instance of domain model [26].

In DSL Tools, DSL Designer is used to define domain models. When a DSL solution is created, Toolbox is used to drag elements to DSL Designer to define domain-specific languages and Properties Window is used to set properties for them. Once the domain model is defined, all templates should be transformed by using the tool in Solution Explorer. Then, designed domain model can be built. Once the domain model is built, a separate experimental build instance of Visual Studio is opened as a generated designer from which diagrams can be created, text templates can be run and domain-specific language can be deployed [30].

The main steps in building a DSL in MS/DSL Tools are:
1. Defining the domain model in .dsl file.
2. Defining the notation elements such as shapes and connectors. The XML serializations of notation elements are automatically generated and stored in a separate file named .dsl.diagram.
3. Defining visualization of domain model via notation elements (mapping shapes to classes and connectors to relationships).

The MS/DSL Metametamodel shown in Figure 6 offers classes, value properties, and relations such as embedding (composition), reference (aggregation) and inheritance [47].

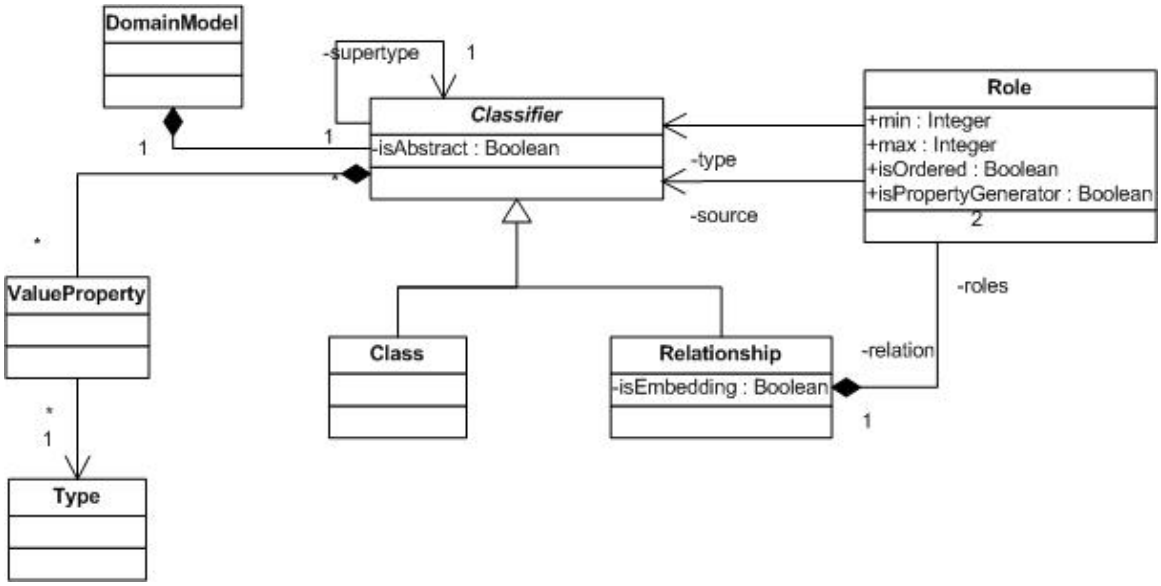

Figure 6 Simplified version of MS/DSL Tools metametamodel [5, 26]

The DSL tools can be used for constructing custom visual designers tailored to any problem domain. For instance, one can create a business process modeling tool using the DSL tools to describe the concepts specific to the way an organization models business processes. If one is building a state chart tool, what a state is, the properties a state has, what kinds of states exist, how transitions between states are defined, etc can be described. On the other hand, the notion of a statechart used to describe the status of contracts in an insurance company and the notion of a statechart used to describe user interaction amongst pages on a Website are similar in concept, but may have very different implementations. Thus, by creating a custom designer, one can specify the exact definition of statechart concepts that are needed [3].

3.3.2 Eclipse Modeling Project

At the beginning, Eclipse was the IBM IDE for Java development. Currently, it is an open source tool platform for many other technologies and projects. The Eclipse Modeling Project unifies the modeling related frameworks, tools and standard implementations. This project includes [32, 33]:

- **Eclipse Modeling Framework (EMF)**: EMF is a modeling framework and code generation facility for specifying metamodels and managing model instances [32]. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, a set of adapter classes that enable viewing and editing model, a basic editor. Models can be specified using annotated Java, XML documents, or modeling tools, then they can be imported into EMF [20]. EMF is closely bound to UML and MOF. EMF includes its own metamodel called Ecore which is used for describing models and runtime support for models. Ecore metametamodel is shown in Figure 7 and it is compliant with MOF. The runtime support includes notification, persistence and XMI serialization [12].

Other fundamental parts of EMF are EMF.Edit and EMF.Codegen. EMF.Edit framework includes generic reusable classes for building editors for EMF models [26]. EMF.Codegen provides code generation facility to build a complete editor for an EMF model. It includes a GUI from which generation options can be specified, and generations can be invoked [26]. The generated editor uses classes from EMF.Edit framework to provide standard table and property sheet views. If the generated editor is not sufficient, it can be customized by hand-coding [28].

Open source meta-tools such as EMF are based on the OMG Meta Object Facility (MOF). It takes an abstract syntax expressed in MOF and generates memory-in-store and serializer for that abstract syntax. The serializer implements the generic metamodel to XML mapping defined by the OMG XML Metadata Interchange (XMI). The weakness of EMF is that it focuses on generating code that integrates with a specific tool, Eclipse. However, it lacks support for well-formed rules and concrete syntax [1].

EMF requires editors to be created for editing models compliant to different metamodels. Editors are being assembled from reusable component, nevertheless the metamodels are not used to dynamically configure the editors, thus much more time is needed to assemble an editor for a given metamodel [12]. Similar to MS/DSL Tools EMF provides template-based approach for code generation based on the JET template engine [25].

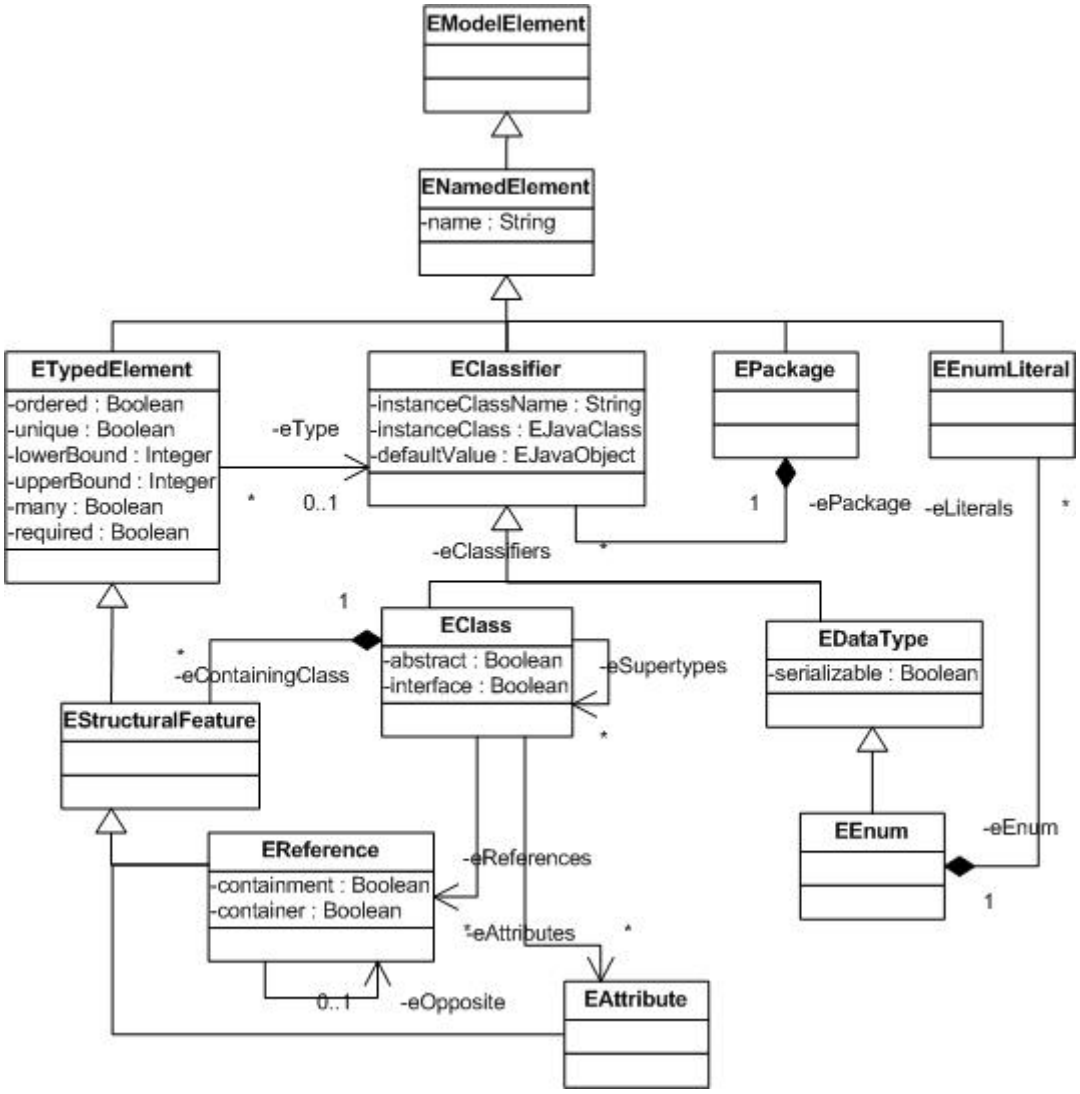

Figure 7 Simplified version of Ecore metametamodel [5]

- **Graphical Editing Framework (GEF)**: GEF allows developers to create a rich graphical editor from an existing application model. GEF employs a Model-View-Controller (MVC) architecture pattern which enables simple changes to be applied to the model from the view [21]. Model can be an EMF model, or something completely different. Although EMF and GEF can be used separately, building DSM support requires both of them. In this sense, GEF provides the graphical support required for building a diagram editor on the top of the EMF framework. [28].

- **Graphical Modeling Framework (GMF)**: GMF provides a generative component and runtime infrastructure for developing graphical editors based on EMF and GEF [22]. GMF functions like a generative bridge between EMF and GEF. MOFScript aims to provide support for model to text transformations [32]. Figure 8 illustrates Doman Model Editor provided by GMF.

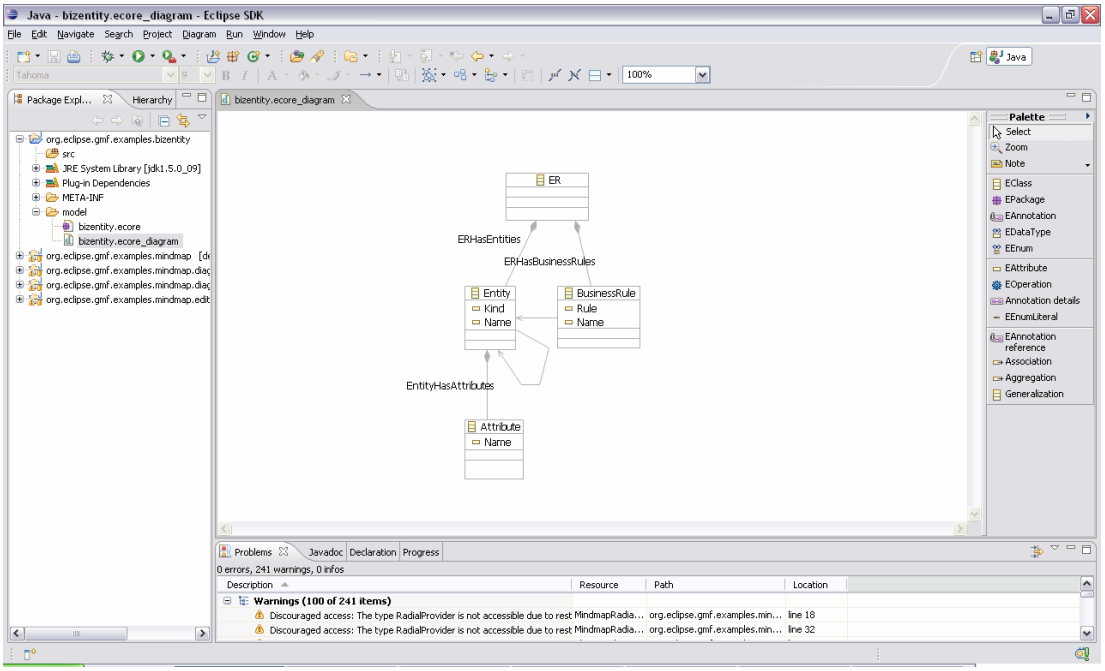

Figure 8 Domain Model Editor provided by GMF

3.3.3 Bridging MS/DSL Tools and EMF

The capability of exchanging models defined using two modeling technologies is a necessity for interoperable MDE platforms. Bezivin et al [26] cite that the capability of exchange models between an EMF and a corresponding MS/DSL Tools based system requires an abstract understanding of both architectures and accurate organization of the interoperability scheme. In order to implement a bridge between MS/DSL Tools and EMF, functionalities provided by the AMMA (ATLAS Model Management Architecture) model engineering platform can be used. AMMA is built on top of the EMF and consists of four different components. The most important component in bridge implementation is a model transformation language named ATL (ATLAS Transformation Language) [26]. An ATL program transforms a source model conforming to a source metamodel into a target model conforming to a target metamodel [77].

The bridge between MS/DSL Tools and EMF spans two levels: metamodel and model level shown in Figure 9. At the level of metamodels, the bridge allows to transform MS/DSL Tools domain models to EMF metamodels. At the level of models, the bridge allows transforming MS/DSL Tools models conforming to domain models to EMF models conforming to EMF metamodels. At both levels the bridge operates in both directions. The benefit of using such a bridge is the capability to exchange the models defined in MS/DSL Tools and EMF platform [26].

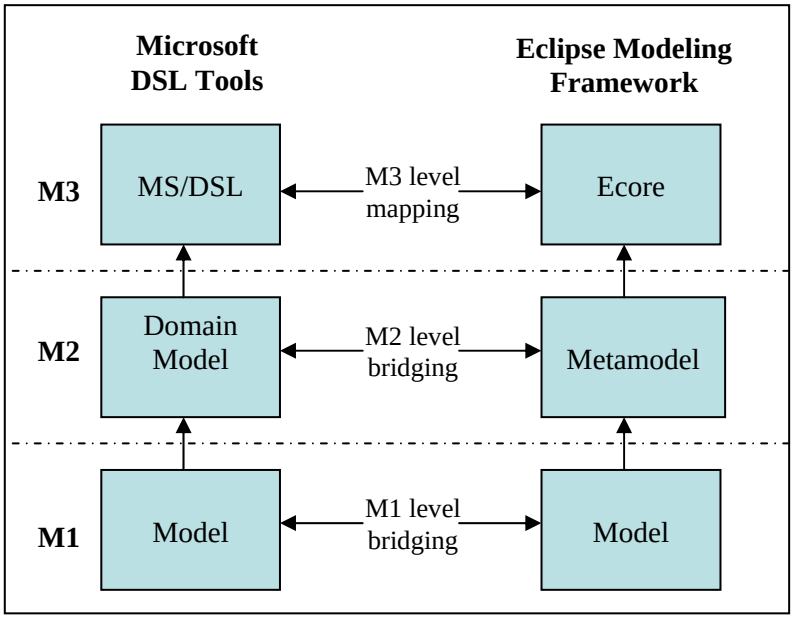

Figure 9 The overview of two level bridging [26]

The bridge performs transformations in two levels: M2 level and M1 level. The M3 level mapping allows establishing correspondences between metametamodels. Since Microsoft does not specify an explicit metametamodel for DSL Designer, the metametamodel of AMMA platform named KM3 (Kernel MetaMetaModel) [77] is used as a mediator between the Ecore and DSL metametamodel. The AMMA platform provides mappings between KM3 and other metametamodels such as Ecore [26].

The M2 level transformation (DSL to Ecore) process includes three steps [26]:
1. **XML2DSL:** A DSL definition file (.dsl) is injected into an XMI file conforming to an XML metamodel, using an XML injector. Then, an ATL transformation is applied to capture information from the XML file to a model which conforms to the DSL metametamodel [26].
2. **DSL2KM3:** The DSL model is transformed to a KM3 metamodel, using another ATL transformation. DSL classes are mapped to KM3 classes, like types and properties [26].
3. **KM32Ecore:** The existing transformation of KM32Ecore provided by AMMA platform is used to get a metamodel conforming to EMF. In that way, this model can be used within any EMF compatible tool such as Omondo's EclipseUML [42] plug-in [26].

In addition, the M2 level transformation (Ecore to DSL) process includes three steps as well [26]: KM32DSL, DSL2XML and XML2Text. In order to implement M1 level bridge, a DSL model file has to be transformed to a model that conforms to a metamodel defined under EMF [26].

Bezivin et al [26] argues that achieving model interoperability is much more complex than simply defining a local serialization format in a local context like XMI. In order to achieve model-based interoperability, using a monolithic language like UML 2.0 is not a solution. Instead, a metalanguage building system like MOF can be used to define a set of DSLs through metamodels like UML, SPEM (Software Process Engineering Metamodel), CWM, and EDOC (Enterprise Distributed Object Computing). Implementing such a bridge between MS/DSL Tools and EMF makes it possible to hide their implementation complexity to the end-users. In this way, end-users will be able to work indifferently (with an XML-schema, an MOF-metamodel or a MS/DSL Domain Model) on solving the real problems without dealing with the artificial problems introduced by fragmented and complex modeling technologies [26].

4 COMPARISON OF MS/DSL TOOLS AND ECLIPSE FOR DOMAIN-SPECIFIC MODELING

4.1 Introduction

In this chapter, the author developed a small Domain-Specific Modeling language from the scratch in both MS/DSL Tools and Eclipse EMF/GEF/GMF. First, different phases in DSL development are described and how to integrate DSM tools to existing development environment is discussed. Later, two DSL designers were built in adherence to the specified small DSL by using both MS/DSL Tools and Eclipse EMF/GEF/GMF. Finally, an evaluation of two modeling tools was performed based on a set of criteria with help of experiences gained in development of two DSL designers.

Since DSLs can be used to specify business requirements, the defined domain-specific language specifies a business entity diagram to capture domain information of a banking application. A Business Entity Diagram shows the entities involved in a business process. Each entity has a set of attributes and entities relate to each other. Relationships are represented by lines between the business entity shapes. Business rule shapes describe how entities are used together. This simple Business Entity language can be used to generate different artifacts such as database definitions, user interfaces for the business entities and skeletons for business logic from the rules [61].

4.2 DSL Development

DSL development requires both domain knowledge and language development expertise. The development process should be facilitated by using DSM tools. Mernik et al [66] cite that DSL development consists of five phases: decision, analysis, design, implementation and deployment. DSL development is not a sequential process meaning that former phases are often influenced by latter phases [66].

4.2.1 Decision

It is costly to design, build and maintain DSLs regardless of which DSM tool (MS/DSL Tools, Eclipse or other tools) is used in development. Thus, in design phase a cost-benefit analysis should be carried out to determine whether it is more beneficial to use DSLs as opposed to general-purpose languages or not.

As it is illustrated in Figure 10, DSL development requires higher startup costs due to complexity of designing and implementing DSLs. However, it reduces development life cycle costs afterwards due to productivity increase [67]. In this sense, investment in DSL development has to pay itself by more economical software development and maintenance in later applications [66]. Once the decision is made towards DSL development, next step would be to define the scope and size of the language and to select the appropriate DSL tools.

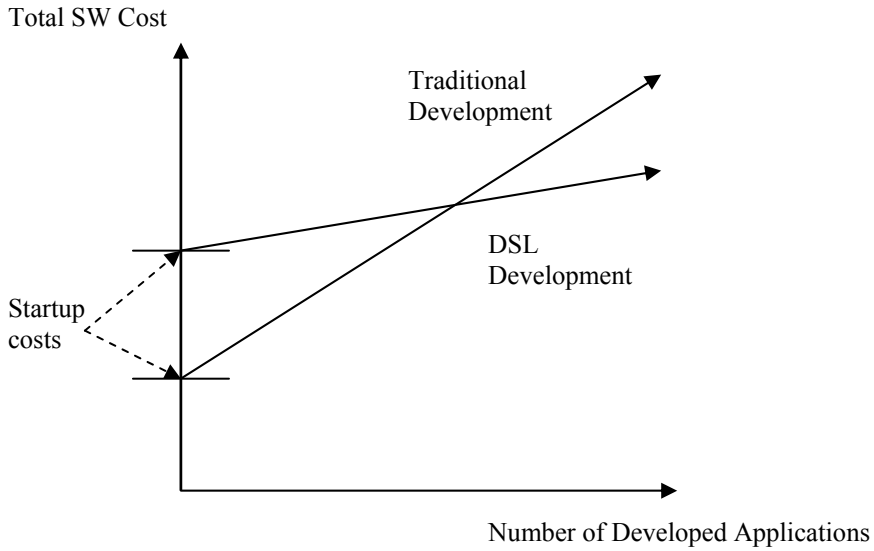

Figure 10 The payoff of DSL Development [67]

4.2.2 Analysis

In analysis phase of DSL development, problem domain is identified and domain knowledge is gathered. Various implicit or explicit sources for domain knowledge could be existing technical documents; knowledge provided by domain experts, existing models defined in general-purpose languages, and customer surveys. In addition, knowledge engineering practices such as knowledge capturing and knowledge representation can facilitate elicitation of domain knowledge. The output of domain analysis should include a domain model consisting of a definition of the scope of the domain, domain terminology, description of domain concepts and commonalities, variabilities and dependencies between them [66]. Völter et al [68] cite that when trying to find an appropriate DSL for a certain domain, one should always take into account the required amount of variability. To express variants in the models, tool support (e.g. feature diagrams) should be available [68].

4.2.3 Design

The easiest way to design a DSL is to base it on an existing language and to extend existing language with new features that address domain concepts. Once the relationship to existing languages has been resolved, a DSL designer should be used to specify design before implementation [66].

Kolovos et al [64] cite that quality attributes of a DSL and its supporting environment (e.g. compilers, IDE) have an impact on the quality attributes of the overall development process and the resulting product. In design phase of DSL development, the quality requirements of the DSL should be specified and necessary trade-offs should be made. The core requirements for a DSL are as follows [64]:

- **Conformity:** The language constructs must correspond to important domain concepts.
- **Orthogonality:** Each language construct is used to represent exactly one distinct concept (e.g. Business Entity, Business Rule) in the domain.
- **Supportability:** DSL tools shall provide support for model management and transformation.
- **Integrability:** It is essential to integrate DSL tools with other facilities used in development process. DSL tools should provide extensibility mechanism to support additional constructs and concepts.

- **Longevity:** In order to ensure tool support and to make it possible to quantify the payoff obtained from using the DSL, the DSL should be used for a non-trivial period of time. It is assumed that the selected software domain would persist for a sufficiently lengthy period of time to justify the cost of building DSL and its tools.
- **Simplicity:** A DSL should be simple as possible in order to express the domain concepts and to support its users.
- **Quality:** The DSL shall provide general mechanisms for building quality systems that include language constructs for improving reliability, security, safety, etc.
- **Scalability:** The DSL constructs should be able to manage large-scale descriptions. However, it is not a necessary requirement, besides small languages are preferable.
- **Usability:** DSL constructs shall be expressive and easy to understand.

4.2.4 Implementation and Deployment

Once an executable DSL is designed, the most appropriate implementation approach (e.g. interpreters, compilers/application generators, embedded) should be chosen. Depending on the selected approach some implementation trade-offs have to be considered. Compilers/application generators provide syntax which is close to the notation used by domain experts, so that they facilitate domain-specific analysis, verification and optimization. However the required effort to design compilers/application generators is large comparing to embedded approach [66]. It is common that generators are used for structural aspects of a system and interpreters for behavioral aspects. The main pitfall of interpreter approach is that interpreters are inherently slower than generated code. On the other hand, the main advantage of interpreters is that they allow late binding at runtime so that a model can be changed at runtime without the need for redeployment or rebuild. In case of generators, generated code is bounded at compile time, so after each modification in the model, regeneration becomes a necessity. Decision regarding whether to build generators or interpreters for a given domain draws the boundaries of the domain, thus decision can be postponed until a solid understanding of the domain is achieved. [68].

Regarding code generation, an important issue is to combine generated code and hand-written code in a controlled environment. In order to avoid inconsistency problems that appear during modification of generated code, developers should be guided by IDE regarding what they are allowed to access in generated code. Another solution could be to keep generated and manually written code in separate files. An architectural description that clearly defines which artifacts are generated and how generated code is combined with manually written code is needed [68]. In the case of MS/DSL Tools, use of partial classes that can be regenerated can maintain separation of generated code and manually written code efficiently. In the case of Eclipse, descriptive tags like “@generated” are used to serve for this purpose instead of using separate files. If handwritten code must be inserted to the generated code, introduction of protected areas is required. These areas can be read by the code generator, in this way manually written code will not be overwritten during regeneration. However, the disadvantages of this approach are that code generator and versioning becomes more complex as well as the separation of generated and non-generated code dissolves [68].

Völter et al [68] argue that quality of generated code depends on the transformations such as templates that generate C# in MS/DSL Tools. Thus, if transformations are created with necessary care, generated code’s quality will not be worse than hand written code, besides it is more systematic and consistent than hand written code [68].

Regarding documentation, the DSL and its semantics must be documented to enable its efficient use in development projects. Although models document the domain concepts in a form that can be understood by domain experts, user manual and other documentation can

also be generated from models. The advantage of generated documentation is that it is only required once for each DSL, not for each application [68].

Regarding deployment, Visual Studio 2005 allows to create DSL Tools Setup Project template, when it is built, a setup.exe file for installing the DSL designer is generated. This setup project automates the process of producing a Microsoft Windows Installer (MSI) package to deploy a graphical designer and its associated component. Due to seamless integration into the Eclipse IDE, graphical editors developed using Eclipse GEF/GMF can directly use Eclipse's code generation and other plug-ins and can be readily packaged (as .jar files) and deployed [74].

To conclude this section, one can say that both MS/DSL Tools and Eclipse provide support for design (metamodeling), implementation and deployment phases of DSL development, however they do not provide any support for decision and domain analysis phases.

4.3 Integrating DSM Tools into Development Environment

Integration of DSM Tools into existing development environments and other MDD tools has a direct affect on the success of DSL development. Allen et al [62] cite that integrating modeling tools is a requirement for integrating verification and validation functions of these tools. Both MS/DSL Tools and Eclipse EMF/GEF/GMF provide validation mechanisms for each diagram element in domain models defined under these tools. When one model is exchanged between two modeling tools by using a bridging mechanism, the transformed model should be a valid model in the target modeling tool as well. In this sense, well-tested exchange of model information between tools reduces the loss of fidelity when the model is transformed. In addition, as modeling work moves from tool to tool, if the transferred model is well-tested, the domain expert does not lose the ability to evaluate modeling quality [62].

MS/DSL Tools is included in Visual Studio 2005 SDK (Software Development Kit) which allows developers to integrate tools, editors, designers, languages etc. inside Visual Studio 2005. Kelly [63] criticizes MS/DSL Tools' tight integration with Visual Studio due to the reason that integration between model elements and IDE behaviour is breaking the low coupling / high cohesion rule which is a traditional design goal. In this sense, it would be difficult to maintain models defined in older releases of DSL Tools and eventually it will prevent the modeling language evolving [63].

Gundy et al [74] describe an approach for generating Eclipse-based multi-view graphical editors for DSL implementation. This approach reuses an existing meta-tool, named Pounamu to specify underlying model and graphical editor, together with Marama, a set of Eclipse plug-ins that extend the Eclipse GEF and EMF to interpret the tool specifications and realize them as a high-quality Eclipse tool. This approach is used to implement a wide range of Eclipse-based DSL tools. These Eclipse-based tools are implemented with high productivity comparing to coding the same tools with the standard Eclipse GEF and EMF. In addition, generated tools have better usability and robustness [74]. Gundy et al [74] mainly aim to generate Eclipse GMF and MS/DSL Tools specifications from their Marama-implemented meta-tools to leverage others' frameworks for building DSL tools. In this way, tool specifications can be used to generate different tool realizations based on the end-user needs and satisfaction [74]. This approach facilitates modification and extension of DSL tools, so that DSL tools can be integrated with less effort to existing development environments.

Völter et al [68] cite that integration of system developed using MDD with existing systems and infrastructures includes the systematic mapping of various APIs, as well as protocol transformations between these APIs. Integration code must be added either to the application to be developed or into the existing legacy application that is to be integrated. In addition, a specific DSL which supports description of the mapping rules between model elements and existing legacy applications should be defined [68]. In this regard, integration of MDD tools with existing systems increases their strength.

4.4 Building a DSL Designer using Microsoft DSL Tools

In this subsection, the author has developed a Business Entity language designer named BEL using September 2006 CTP release of MS/DSL Tools included in Visual Studio 2005 SDK version 3.0. DSL design process using MS/DSL Tools includes the following steps shown in Figure 11 [30]:

1. Running the wizard creates and configures the domain-specific language solution from provided templates such as Class Diagrams, Component Models, Task Flow or Minimal Language. As a result of running the wizard, the created solution contains two projects: Dsl and DslPackage. The Dsl project defines the domain-specific language including its diagram, domain classes, relations and its editing tools. DslPackage project determines how the language tools match to Visual Studio. DslPackage allows users to add new functionalities to Visual Studio. These functionalities are related to language's features like its new file extension, its toolbox, and its graphical designer.
2. Creating domain-specific language includes defining classes and their relationships, and diagram elements such as shapes and connectors.
3. Code (.cs files) is generated from text templates (.tt files) which read the language definition. Language definition can be customized in DomainModel.tt file. Whenever language definition files are altered, all templates should be transformed by clicking the Transform All Templates button on the Solution Explorer toolbar.
4. Debugging the solution (F5) launches an instance of Visual Studio experimental build with a debugging project which can be used to test the domain-specific language. Text templates can be built within the debugging project, which reads the model that is created by using domain-specific language and generates code and other artifacts from it.
5. After verifying the domain-specific language, a deployment package (.msi file) can be created to distribute the domain-specific language.

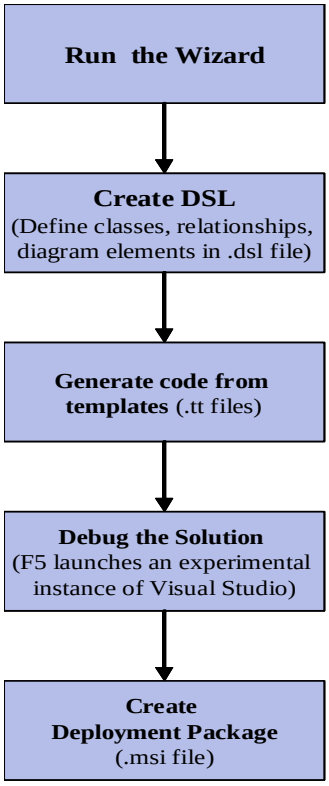

Figure 11 Domain Specific Language Design Process [30]

4.4.1 Building the Domain Model for Business Entity Diagram

Building a domain model starts with defining its structure including domain classes and their relationships. The structure of BEL domain model is presented in Figure 12. Table 4 summarizes the domain classes and their relationships defined by the author:

- **Domain Classes:** Entity Relationship (ER), Entity, BusinessRule, and Attribute. They represent basic elements of the DSL. Each class refers to a domain concept. In addition, domain properties such as Kind and Name properties for Entity Class, Rule and Name properties for BusinessRule Class are defined.
- **Embedding Relationships:** ERHasEntities, ERHasBusinessRules, and EntityHasAttributes. Each embedding relationship represents a strong relationship between two domain concepts. For instance, an Entity Relationship has Entities and BusinessRules, an Entity has Attributes.
- **Reference Relationships:** EntityReferencesEntities and BusinessRuleReferencesEntities. Each reference relationship represents a weak relation between two domain concepts. Multiplicities (one to one, one to many, many to many etc.) for each type of relationship are also specified. For instance, one Entity may reference more than one Entity; one BusinessRule may reference more than one Entity.

Domain Class	Relationship (Embedding/Reference)	Domain Class
ER	ERHasEntities	Entity
ER	ERHasBusinessRules	BusinessRule
Entity	EntityReferencesEntities	Entity
Entity	EntityHasAttributes	Attribute
BusinessRule	BusinessRuleReferencesEntities	Entity

Table 4 Domain classes and relationships between them

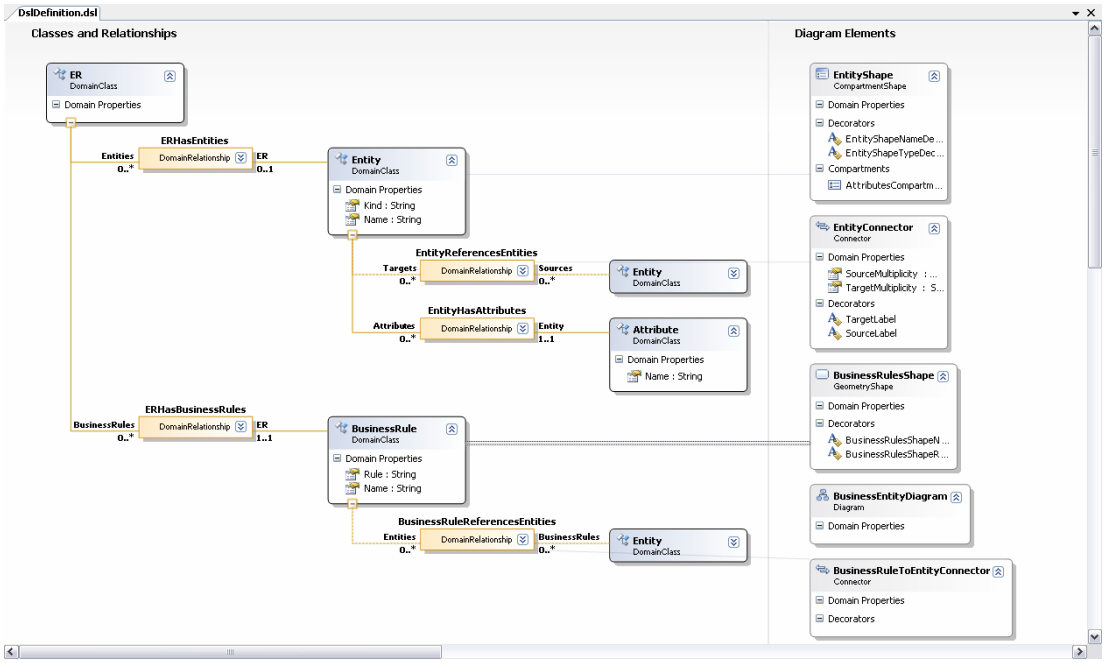

Figure 12 The structure of domain model for Business Entities Diagram in MS/DSL Tools

4.4.2 Describing the Graphical Notation

After defining domain classes and their relationships, the next step is to describe graphical notation for diagram elements. This notation includes shapes and connectors in the toolbox of the resulting DSL designer for BEL. Entity shape is defined as it is including attributes in a compartment. In addition, for each shape or connector a set of text decorators are defined. Text decorators' names correspond to domain properties' names of classes and domain properties' names of connectors which will be displayed in the resulting DSL designer.

4.4.3 Mapping the Graphical Notation to the Domain Model

After describing each graphical notation elements, necessary mappings are defined including mapping shapes to domain classes and connector lines to domain relationships. In this way, when users create shapes and connectors in the BEL designer, they also create the corresponding classes and relationships in the domain. Table 5 summarizes shapes and connectors and their mappings to the domain classes and relationships. In addition DSL Details window is used to map decorators and compartments to the shapes and connectors.

Shapes and Connectors	Map Type	Class/Relationship
EntityCompartmentShape	CompartmentShapeMap	Entity
EntityConnector	ConnectorMap	EntityReferencesEntities
BusinessRulesShape	ShapeMap	BusinessRule
BusinessRuleToEntityConnector	ConnectorMap	BusinessRuleReferencesEntities
AttributesCompartment	CompartmentMap	Entity

Table 5 Diagram elements and their mappings to the domain classes and relationships

4.4.4 Running the Business Entity Language Designer

Once the domain model and its graphical notation including mappings are defined, the source code for the BEL designer can be generated by transforming all template files. After this step, the Visual Studio BusinessEntity Solution can be built. Building solution opens a

new experimental instance of Visual Studio with a debugging project. This project includes a designer file with an extension named ".be". This extension is dedicated particularly for BEL, which enables separating different DSL designers. Figure 13 shows the toolbox items that are special for BEL designer in the debugging project. The author has built a Business Entity Model shown in Figure 14 by using BEL designer in the debugging project. This business entity model includes the domain concepts of a banking application. These concepts include entities, business rules as well as relationships between them. Based on personal experiences, the author has selected some typical business entities in a banking application such as Bank, Branch, Account, Loan, Customer and Transaction. As it is specified in domain model of BEL designer, a business rule represents how two entities relate to each other. For instance, the TransferRule represents how Account and Transaction entities relates in a way that business rule name states under which condition two entities can interact: "a source account must be in credit to be able to confirm a payment transaction". In addition, each domain entity has a set of attributes which characterize the data fields that are particular for each entity. For instance, Customer entity has SSN, Name, Address and Phone attributes. Furthermore, source and target multiplicities of domain relationships is specified to provide better understanding regarding how domain concepts interact with each other. For instance, a customer may have more than one bank account; a bank may have more than one branch etc.

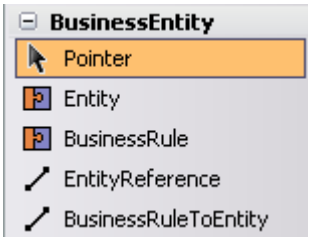

Figure 13 Toolbox support for BEL designer

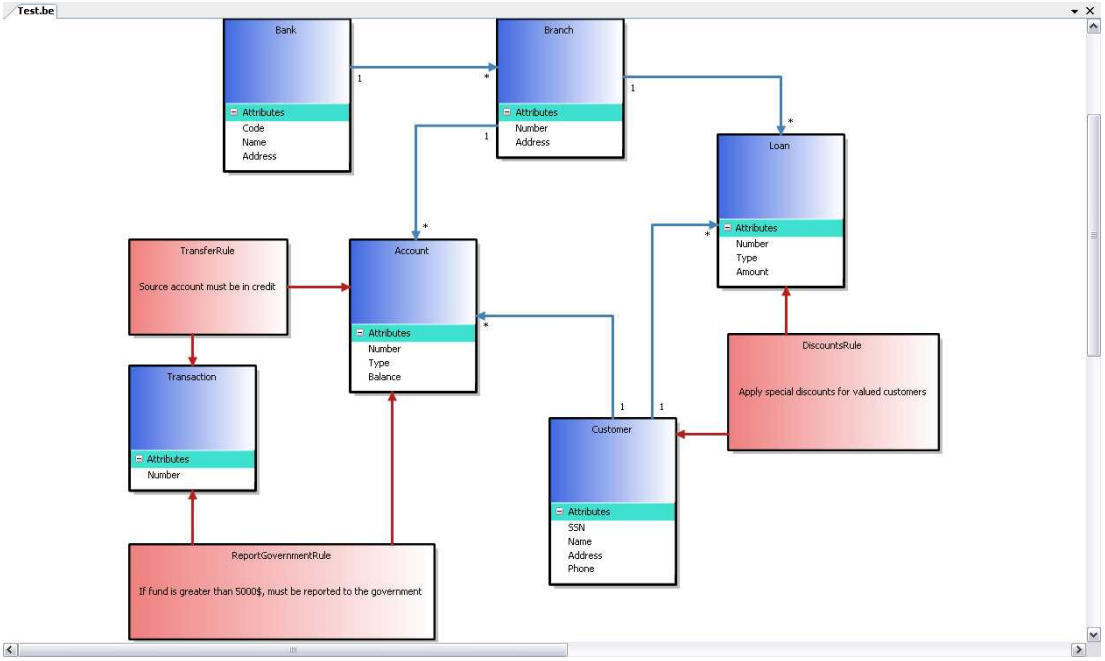

Figure 14 A business entity model built using BEL Designer in MS/DSL Tools

4.5 Building a DSL Designer using Eclipse EMF/GEF/GMF

In this subsection, the author has developed a Business Entity language designer using Eclipse Project SDK version 3.2.0, EMF version 2.2.0, GEF version 3.2.0 and GMF version

1.0.0. One can generate graphical editors for domain models by using either EMF/GEF-based editors or GMF-based editors. In GMF-based editors, creating a graphical definition and a mapping definition to the chosen domain can be used to generate much of the basic functionality expected in an EMF/GEF-based editor [59]. In this sense, GMF saves a lot of development time and effort comparing to building same editors in GEF. On the other hand, if the domain model does not require developing functionally rich editors, if only static nodes and connections are sufficient, GEF would be the better choice. Nevertheless, since GMF aims to provide a bridge between EMF and GEF, the author has selected GMF-based editors to develop graphical editor for our domain model.

Figure 15 illustrates the main components and models used during GMF-based development:

- Domain Model (.ecore) is developed by defining an Ecore model with EMF Editor or importing the model from other Ecore suppliers. Diagram definition (.ecore_diagram) is maintained separate from the domain model in GMF. In addition, EMF Generator Model (.genmodel) should be created from Ecore model using EMF Model Wizard.
- Graphical Definition (.gmfgraph) is developed by using GMFGraph Model wizard to generate a graphical definition model from existing ecore model. Graphical definition model contains information related to the graphical elements (figures, nodes, compartments, links, etc.) that will appear in a GEF-based runtime; however have no direct connection to the domain models for which they will provide representation and editing [59].
- Tooling Definition (.gmftool) is developed by using GMFTool Model Wizard to develop a tooling definition model to provide tooling support for graphical definition. The tooling definition model is used to design the palette and other periphery such us menus, actions and toolbars [59].
- Mapping Model (.gmfmap) is developed by binding three models: the domain, the graphical definition and the tooling definition via GMFMap Model Wizard. In this way, GMF aims to allow the graphical definition to be reused for several domains. A separate mapping model is used to link the graphical and tooling definitions to the selected domain model(s) [59].
- Once the appropriate mappings are developed, a Generator Model (.gmfgen) -similar to EMF genmodel- is created from mapping model to allow implementation details to be defined for the code generation phase [59].
- Finally, the Diagram Plug-in is generated and then tested in a new Eclipse Application at runtime. The editor plug-in based on the generator model targets a final model, the runtime notation model. When a user is working with a diagram, the runtime bridges the notation and domain model(s) and provides the persistence and synchronization of both [59].

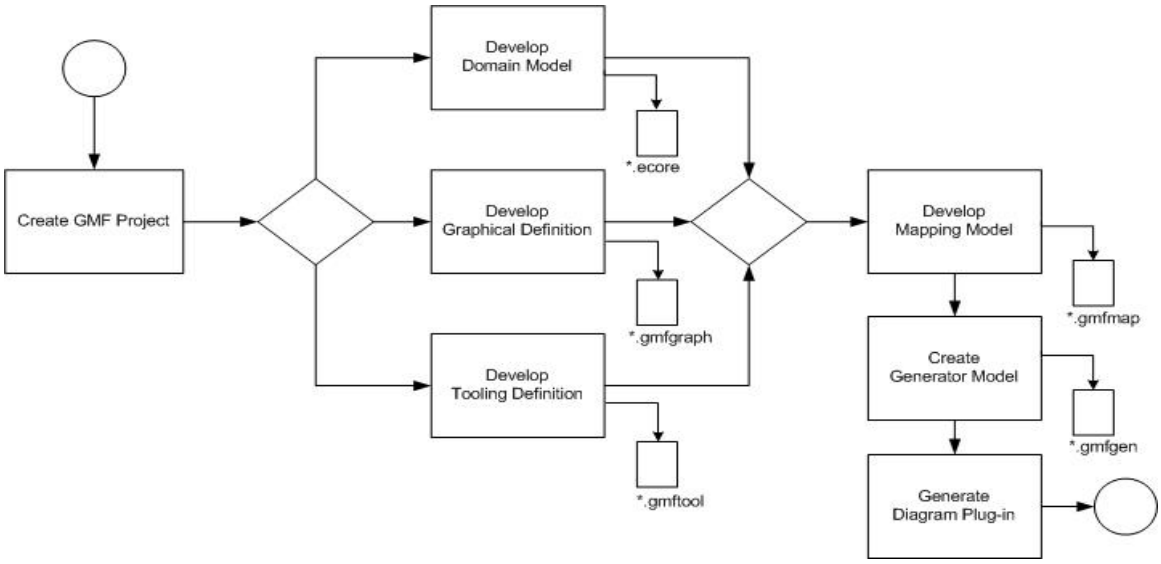

Figure 15 Overview of GMF [59]

4.5.1 Developing the Domain Model for Business Entity Diagram

Although graphical definition is maintained separate from the domain, it is likely that developing a DSL designer in Eclipse GMF starts with developing the domain model. GMF provides a graphical editor to complement EMF generated editors. EMF includes its own metamodel called Ecore which is used for describing the domain model for business entity diagram. Since EMF is closely bound to UML and MOF, the author defined a set of UML-like Ecore elements which correspond to domain concepts and their relationships shown in Figure 16:

- EClasses are ER, Entity, BusinessRule, and Attribute.
- Aggregation relationships are ER contains Entities, ER contains BusinessRules and Entity contains Attributes.
- Association relationships are Entity to Entity and BusinessRule to Entity.

Once the Ecore model is defined and diagram file is initialized, a generator model is defined. Then, the domain model and its edit code are generated from the generator model.

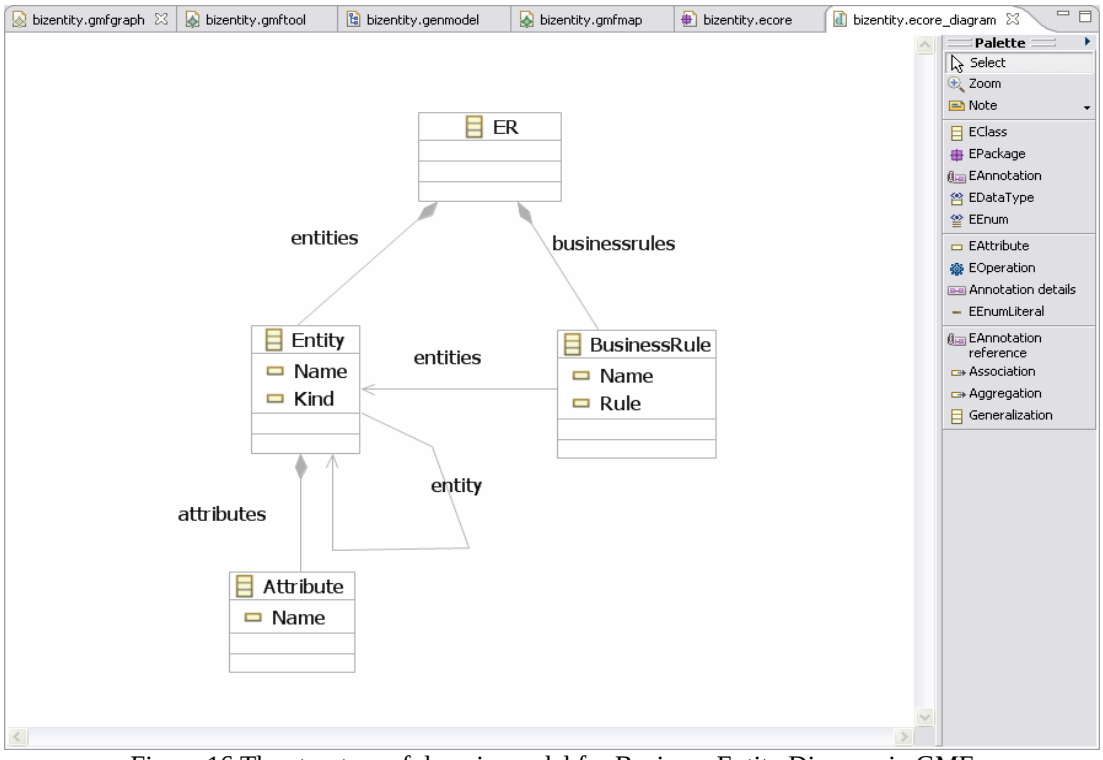

Figure 16 The structure of domain model for Business Entity Diagram in GMF

4.5.2 Developing Graphical Definition

Once the domain model and edit code is generated, next step is to develop graphical definition model for business entity diagram. Similar to diagram elements of domain model in MS/DSL Tools, a graphical definition model is created including figures, nodes, compartments and links. The author defined the following graphical elements:

- Nodes: Entity, BusinessRule and Attribute.
- Connections: EntityEntity and BusinessRuleEntities.
- Figures: Rectangle for Entity and Rounded Rectangle for BusinessRule, Polylines for EntityEntity and BusinessRuleEntities connections.
- Compartment for Attributes.

4.5.3 Developing Tooling Definition

Once the graphical definition model is developed, next step is to develop tooling definition model. The tooling definition is used to specify the palette, creation tools, etc. for graphical elements defined in previous step. Thereby, the author defined palettes and creation tools for nodes (Entity, BusinessRule and Attribute) and links (EntityEntity and BusinessRuleEntities) in separate groups.

4.5.4 Developing Mapping Definition

After developing tooling definition model, the author developed mapping definition model which binds the three models developed so far: the domain, the graphical definition and the tooling definition. Despite the manually defined Label Mappings for each node, node and link mappings are automatically generated from existing three models shown in Figure 17. Then, mapping definition model is used as an input to transformation step which produced the final model, the generator model. After generator model is created from mapping model, all diagram code is generated from the generator model.

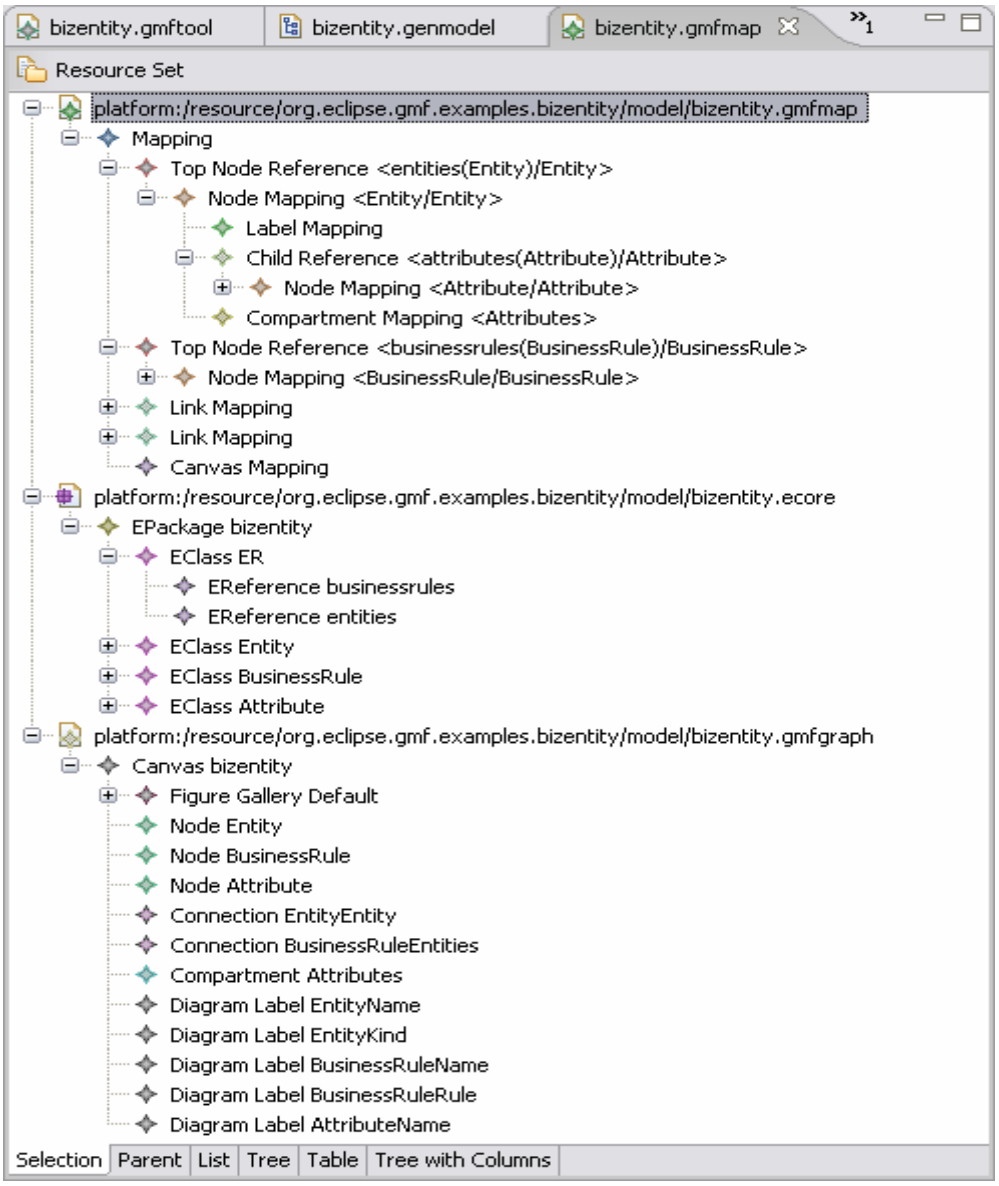

Figure 17 Mapping Definition Model in GMF

4.5.5 Running the Business Entity Language Designer

Once the required plug-in is generated, a new Eclipse runtime workspace is launched to test the DSL designer for Business Entity diagram. Similar to the business entity model which was built in MS/DSL Tools, the author defined the same entities, business rules and relationships using DSL designer in Eclipse. Figure 18 shows the business entity model for a banking application which is defined using Business Entity Language designer in Eclipse.

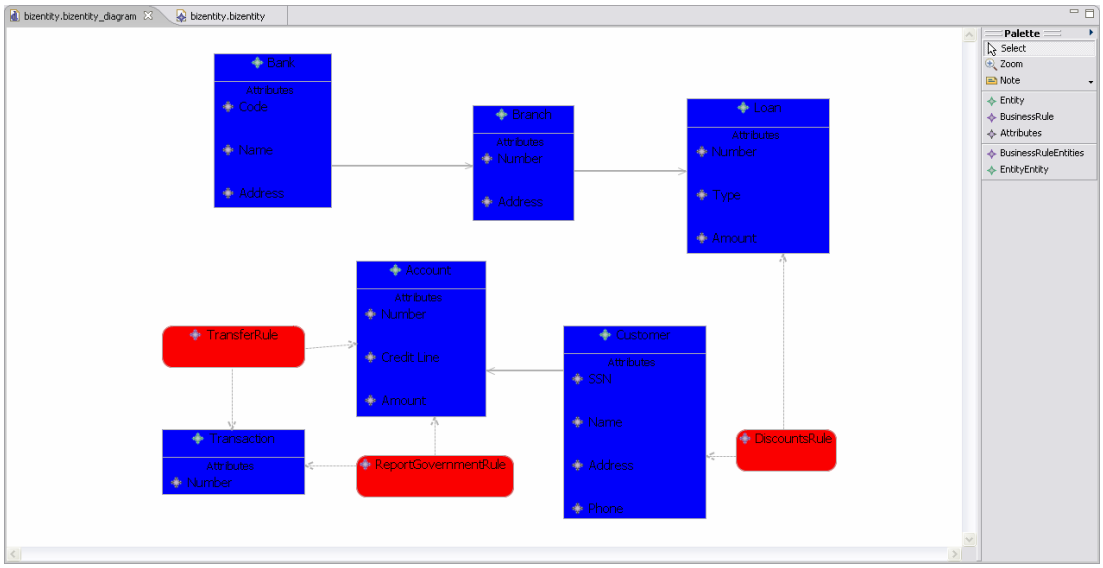

Figure 18 A business entity model built using BEL Designer in Eclipse

4.6 Comparison

In this section, the author has identified a set of criteria that are taken as basis to evaluate MS/DSL Tools and Eclipse Modeling Project. Pelechano et al [32] discuss the differences between two modeling technologies based on a set of criteria in Table 6. The author has taken these criterions as basis for comparison and preparing guidelines in this thesis.

Criteria	MS/DSL Tools	Eclipse
Metamodeling	Proprietary Notation (domain model)	EMF (Ecore)
Repository	XML File	XML, XMI
Graphical notation	Direct Manipulation of XML files	GMF
Model to Model	No explicit technique	ATL, etc.
Model to Text	Proprietary template language	MOFScript, etc.
Model Validation	Programmatic validators	Validation Framework (EMF Technology Project)

Table 6 DSL Tools vs. Eclipse [32]

Regarding metamodeling, MS DSL Tools provide a proprietary notation and a graphical environment for specifying the language metamodel. A DSL model conforms to a domain model which is the Microsoft term for metamodel. On the other hand, EMF provides a complete metamodeling and model management environment, which uses Ecore as language for metamodel specification. Regarding serialization of models or model repository, MS/ DSL Tools provide a XML proprietary format, whereas EMF support XMI standard or any XML schema-format defined by the users [26, 32].
For the definition of graphical editors, MS DSL Tools provide a graphical mechanism including the direct manipulation of XML files. GMF provides wizards, cheat sheets and

other utilities for the definition of graphical editors whereas MS/DSL Tools provides walkthroughs and online documentation [32].

MS/DSL Tools lack an explicit technique for supporting model-to-model transformations, whereas Eclipse community proposes Model-to-Model Transformation (M2M) project which will provide an implementation of emerging OMG standard for model transformation, Query-View-Transformation (QVT) under the Eclipse Modeling Project. Currently, many projects (ATL, Viatra2, etc.) exist for model-to-model transformations which manipulate Ecore models. Model transformation languages like VIATRA (VIsual Automated model TRAnsformations) and ATL have different levels of compliance to the QVT standard [32, 34]. Regarding model-to-text transformations (code generation), MS/DSL Tools provide a proprietary template language which is integrated in the environment. This language enables injection of C# or VB source code. In the Eclipse, all the Java-based template languages or MOFScript plug-in can be used for the model-to-text transformations [32].

When developing a DSL, there is a need to add constraints (e.g. an entity name property must not contain spaces) to domain models for checking their semantic correctness beyond what is expressed in the syntax of that language. MS/DSL Tools allows using partial classes in which methods are defined programmatically for each constraint. There partial classes are implemented for the class of the domain model that requires validation. The constraint method uses the property, the relationship and the role names that are defined in the domain model [73]. On the other hand, the Eclipse Modeling Framework Technology project includes validation component which provides capabilities used to ensure model integrity. There capabilities include constraint definition which provides API for defining constraints for any EMF metamodel, support for parsing the content of constraint elements defined in specific languages, etc. [72].

As Table 7 illustrates, MS/DSL Tools and Eclipse EMF/GEF/GMF have corresponding DSL design steps. DSL design is consisted of five steps from definition of domain model to code generation. In Eclipse, each design step produces an output file which is particular for only this step. In MS/DSL Tools, modifications in different steps are handled in only two files, DSL definition file and its diagram file.

	Step	**MS/DSL Tools**	**Eclipse EMF/GEF/GMF**
1	Domain Model Definition	.dsl file is created	.ecore and .ecore_diagram
2	Graphical Definition	.dsl.diagram file is created	.gmfgraph file is created
3	Tooling Definition	Editor tools are defined via DSL Explorer window	.gmftool file is created
4	Diagram Element Mappings	Mappings are defined via DSL Details window	.gmfmap file is created
5	Code generation	All text templates (.tt files) should be transformed after each modification	.gmfgen file should be recreated after each modification in mappings

Table 7 Comparison of DSL design steps

4.6.1 Results

Based on the author's experiences gained during development of DSLs using both modeling tools, the following guidelines have been prepared:

- Metamodeling support provided by both MS/DSL Tools (via proprietary notation) and Eclipse EMF (via Ecore) for domain model definition is easy to understand and expressive enough.
- Both MS/DSL Tools and Eclipse GMF clearly separate the model data from its graphical representation.

- MS/DSL Tools metamodel designer is more usable than the EMF metamodel designer. Because, MS/DSL Tools provides a visual designer in which both domain model including classes and relationships and diagram elements and their mappings can be defined. However, Eclipse GMF currently provides visual designer for only domain model definition, a hierarchical tree-view which is less usable is used for developing graphical, tooling and mapping definitions.
- Considering their current functionalities generated graphical designers provided by MS/DSL Tools and Eclipse GMF are easy to use. However, further mechanisms are needed to increase their maturity to the same level as UML diagram designers.
- Regarding code generation, both MS/DSL Tools and Eclipse EMF provide template-based approach, thus developing code generators has medium complexity. However, writing these text templates that generate C# code is a tedious task especially in the absence of supporting tools like Intellisense feature. Thus, template based approach of MS/DSL Tools for generation of artifacts is arguable [50].
- MS/DSL Tools provide reflection only at C# level, reflection at model level is missing. In addition, more support is needed for combining and reusing models [50].
- Both Eclipse GMF and MS/DSL Tools metamodel and APIs are not stable enough for use in production quality code. Tools are still under development, radical changes which cause major refactoring may occur [50].
- MS/DSL Tools lack an explicit technique for model-to-model transformations whereas Eclipse supports model-to-model transformation languages like ATL. Besides, Microsoft's approach aims to generate code from models directly which is better option particularly in large-scale systems where all high and low level models have to be maintained and kept consistent.
- Eclipse provides a more comprehensive model validation framework comparing to programmatic validators of MS/DSL Tools.
- Both Eclipse GMF and MS/DSL Tools enable to customize DSL development tasks such as adding custom attributes to a domain property of a class, using custom tools rather than text templates for code generation. Nevertheless, customization is still a challenging task.

5 RELATED WORK

The goal of this chapter is to look into related work that has been done in the research community and to discuss how these works correspond to the thesis work.

Pelechano et al [32] performed an empirical comparison of Eclipse Modeling Plug-ins and Microsoft DSL Tools. In order to evaluate both modeling tools, an experiment was conducted with 48 last year undergraduate students of computer science engineering. Experiment design included two groups of students for developing a DSL where each group was using a different tool. At the end of the experiment, the students answered several questions about their experiences. Focus groups including lecturers and researchers were used to collect which research questions should be formulated to the students in the survey. The questions of survey focused on qualitative characteristics of MS DSL Tools and Eclipse Modeling Plug-ins that emerged from focus group sessions [32]. The research questions that were formulated during focus group sessions were structured on five categories [32]:

- *Metamodeling:* Questions regarding the understandability and expressivity of the metamodeling technique and the usability of the mechanism provided for the metamodel specification.
- *Graphical Editor:* Questions regarding the capabilities for defining the graphical editor (in the sense of usability and extensibility) and questions regarding perceived quality and completeness of generated graphical editor.
- *Code Generator:* Question regarding the perceived complexity of achieving the implementation of the transformations and preferred technique: a common programming language or a template language (or in general a specialized model-to-text technique).
- *Satisfaction:* Questions regarding the utility of the employed tools and the fidelity to the tools.
- *General Questions:* Questions regarding the availability of the documentation and maturity of one tool to the other.

Pelechano et al [32] draw the following conclusions based on the analysis of the data obtained from the questionnaires:

About Metamodeling:

- EMF and Ecore are easier to understand than the proprietary notation of MS/DSL Tools.
- Ecore is expressive enough to build language models.
- The EMF metamodel designer is more usable than DSL Tools metamodel designer.

About the Graphical Editor:

- Both GMF and DSL Tools Graphical Designer are difficult to use.
- Both generated graphical editors are incomplete and needed to be improved to provide more mechanism for defining and producing professional graphical designers.
- GMF reaches a high degree of acceptability compared to DSL Tools.

About Code Generation:

- The task of defining/implementing a code generator has medium degree of difficulty in both cases; however DSL Tools seem to be more difficult.
- Eclipse users prefer MOFScript to build the code generator, whereas DSL Tools users prefer a different programming language to the template language that is provided by the tool.

About User Satisfaction:

- Both tools are very useful and can be used in future projects.
- Eclipse users are %100 faithful to this environment; however 60% of the DSL Tools user would migrate to Eclipse.

About the General Issues:

- More documentation should be published in both cases.
- Eclipse Modeling Project is more mature and robust comparing to DSL Tools.

Since EMF is published in 2003, Pelechano et al [32] cite that it is more simple, robust and stable than the metamodel editor provided by MS/DSL Tools. When Pelechano et al [32] performed this comparative qualitative study, the last version of MS/DSL Tools was February 2006 CTP. When this thesis was written, Microsoft released version 3.0 of Visual Studio 2005 SDK which allows developers to integrate tools, editors, designers and languages into Visual Studio 2005. This release includes September 2006 CTP of DSL Tools version 1.0. Thus, one can expect more maturity from new version of MS/DSL Tools. Evaluation results of this thesis indicate that MS/DSL Tools and Eclipse become more mature over time. Therefore, the experimental comparison performed by Pelechano et al [32] does not represent current state-of-the-tools especially for MS/DSL Tools.

Warmer and Kleppe [40] present their industrial experiences in building a software factory by using MS/DSL Tools to define multiple small DSLs and multiple small models per DSL. UML does not support defining references between different models and it assumes that developers always work on one large model. Thus, companies that apply model driven development on a large scale are facing problems in managing very large models in terms of file size. How partial models can be used to build large, complex applications is investigated based on the industrial experience of building a model driven Software Factory using multiple DSLs. This Software Factory is targeting web-based, administrative applications and four different DSLs are developed to define partial models based on the application architecture. One partial model has the same characteristics as one source code file. Each DSL can be used in isolation of other DSLs. In this sense, one DSL can be replaced by another one in the software factory [40]. Furthermore, Warmer and Kleppe [40] argues that UML is not a necessity in MDA approach and apart from Software Factories using DSLs fits into MDA approach as well. Number of levels like the PIM-PSM-Code levels in MDA can be increased to facilitate model transformations [40]. The importance of the work of Warmer and Kleppe [40] in context of this thesis is that it indicates that even if MS/DSL Tools lacks an explicit technique for model-to-model transformations, partial classes can be used to generate a model from another model. The drawback of this approach is the complexity that arises from defining and maintaining references between partial models.

Kelly [28] performed a comparison of Eclipse EMF/GEF and MetaEdit+ [44] for domain-specific modeling. MetaEdit+ is the most widely-used commercial metaCASE tool, its first version realized in 1995 (current version is 4.5). In addition to its CASE editing functionality, MetaEdit+ also includes XML import and export, an API for data and control access and a generic code generator. The code generator uses a DSL that allows DSM developer to specify how to walk through models along with other text. Eclipse frameworks and MetaEdit+ offer different approaches for defining DSM support. EMF provides a measure of non-graphical modeling into existing Eclipse development environment, whereas GEF is a good basis for building a graphical editor that does not follow the pattern of CASE tool behaviour. Implementing CASE tool functionality with GEF requires significant amount of coding which is the major disadvantage of GEF. MetaCASE tools allow DSM developers to concentrate on modeling language rather than having to implement editing functionalities. On the other hand, modeling frameworks allow building specific behaviors. Thus, both EMF/GEF and MetaEdit+ can be used to build DSM support depending on the situation [28]. However, when Kelly [28] performed this comparison, GMF was still under development. Thus, the significant time and effort savings provided by GMF in development of graphical editors was not considered in this research.

Ke and Sierszecki [46] compare another meta-modeling tool named Generic Modeling Environment (GME) with Eclipse Modeling Frameworks (EMF/GMF/GEF) and MetaEdit+. GME is a configurable toolkit for creating domain-specific modeling and program synthesis environments. The configuration is accomplished through metamodels specifying the modeling language of the application domain [45]. Eclipse EMF/GMF/GEF provide a model-driven approach for creating domain-specific models from their metamodels and a

flexible graphical environment for editing these models, however developing such a graphical editor is labor-intensive task. MetaEdit+ provides a Symbol Editor that facilitates customization of model visual effects and a code generation tool for easy automatic synthesis and documentation. However, metamodeling process in MetaEdit+ is more complicated comparing to GME or Eclipse EMF. In addition, only the cardinality constraints of relationships are supported in MetaEdit+, whereas OCL is not implemented. In contrast, GME offers both fully implementation of OCL and a powerful metamodeling capability by providing a number of unique metamodeling concepts such as sets, references and aspects [46].

6 DISCUSSIONS

To summarize how MDD works the following steps can be considered: first, developers define models based on the metamodels expressed using a DSL, later using code generation templates, the models are transformed to executable code, at this point if it is necessary (most of the times it is), the generated code is merged with manually written code. In addition, model-to-model transformation technologies may precede code generation [68].

If it is applied correctly, MDD can make developer's work much easier by avoiding redundant code and enhancing software quality through the use of formalized structures [68]. By introducing Software Factories proposal, Microsoft aims to enable software companies to produce customized applications by automating routine development tasks. Software Factories makes the use of DSM by utilizing Domain-Specific Languages (DSLs) for modeling various aspects of an application. On the other hand, OMG's MDA proposal can also support DSL development via MOF and UML Profiles; however there is a lack of quality of DSL developed by using MOF and UML Profiles due to the reason that MOF and UML Profiles lack the ability to define an application correctly, completely, concisely, and conveniently. Nevertheless, MDA does recognize the need for DSLs, because it mainly focuses on one-off development. Due to the cost of developing a DSL and a code generator, Software Factories approach can only be recommended for Product Line Development in which the overall costs are distributed among all members.

In comparison to general-purpose languages, DSLs provide significant gains in expressiveness and ease of use in their domain of application [66]. In this sense, DSLs facilitates separating technological and application specific aspects. Before introducing DSL development to an organization, there is a set of issues that are needed to be addressed: technological, organizational and social issues [65]. Technological issues include introducing and integrating DSL development tools into existing development environments. Furthermore, introduction of DSL development in an organization requires introduction of new roles and responsibilities. Introducing DSL development in large projects with many stakeholders can be difficult due to disagreement of stakeholders on one set of domain concepts. Regarding social issues, instead of putting someone out of job, the skill and competences of developers can be migrated to DSL development.

Once organizations made decisions towards DSL development, selection of appropriate DSM tools becomes an issue. An additional problem in selection of DSM tools is that these tools must compete in terms of usability and quality with the highly polished commercial and open source general-purpose modeling tools [74]. In large organizations where a variety of products are being developed using different development paradigms, there is a need for interoperable development teams which employ both MDD and traditional development methods. Both MS/DSL Tools and Eclipse provide support for deployment of DSL designers which facilitates utilizing these designers in development of particular products.

A domain expert of a banking application can use the Business Entity Language (BEL) designer which is developed in this thesis to express his domain knowledge at a higher level of abstraction. In this sense, domain experts should be included in the development of DSLs as a target class due to the reason that their opinions regarding how and why they want to use a DSL designer are valuable inputs. Domain experts can provide feedback about applicability and ease-of-use of the DSL. In addition, metamodels for a domain have to be continuously improved and validated with cooperation of domain experts and development team. During discussions between stakeholders, errors and inconsistencies in the metamodel can be easily uncovered. This is an advantage of DSLs in comparison to UML models which require UML experts. If something can not be expressed with the metamodel's vocabulary, it

means metamodel has some errors or it is imprecise [68]. On the other hand, including domain experts as a target class may require trade-offs between quality, timeliness, efficiency, reliability, robustness, testability, and usability for other user classes [62].

An important issue in generating code from models is to keep models and source code in sync. Whenever domain models are changed, the source code should be regenerated by either transforming all text templates in MS/DSL Tools or creating a new generator model in Eclipse GMF. Furthermore, manually added code should be maintained separated from the generated code and also should not be affected by regeneration. Both MS/DSL Tools and Eclipse are able to address these issues in code generation. MS/DSL Tools has one advantage over Eclipse due to the use of partial classes. Partial classes are used to add manual code without touching the generated code file, manual code is written in a separate file. On the other hand, in Eclipse predefined tags are used to separate generated code from handwritten code, both generated and manual code are stored in the same code file.

6.1 Threats to Validity

6.1.1 Conclusion Validity

In this thesis the author faced with the challenge to learn how to use both DSM tools and their metamodeling language notations before starting development of DSL designers. The tool documentations and user manuals facilitated learning process significantly. The author had previous personal experiences in using Microsoft Visual Studio development environment, whereas did not have any experience in Eclipse. To minimize this threat the author put more focus on learning Eclipse development environment. The author has spent much more effort in learning how to develop a DSL designer using Eclipse Modeling Frameworks comparing to MS/DSL Tools. During learning process, the author might have overlooked some functionality of the tools which can have an impact on the evaluation results. This can pose a threat to the validity of conclusions regarding ease of use and functionality of graphical editors developed using both DSM tools.

6.1.2 External Validity

The external validity is concerned with the ability to generalize the results of the comparison of two DSM tools. Since this thesis does not focus on any particular domain for the comparison of selected DSM tools, the evaluation results that are based on the author's experiences gained in development of two DSLs can be taken as basis for further evaluation. However, since current versions of both DSM tools provide limited visual notations and editor functionality for generating graphical editors, the author suggests repeating the comparison of MS/DSL Tools and Eclipse EMF/GEF/GMF using more stable versions of these tools focusing on different domains in the future.

7 CONCLUSIONS

In this thesis, the author has presented a comparison of Microsoft DSL Tools and Eclipse Modeling Frameworks (EMF/GEF/GMF) for Domain-Specific Modeling (DSM) in the context of the Model-Driven Development (MDD). First, the author has performed a literature survey of existing MDD and DSM standards and principles. The current state-of-art in MDD and DSM indicates that more mature DSM tools that better conform to MDD standards are needed. Since MDD approaches (MDA, Software Factories, MIC, etc.) are still under development, DSM tools should be continuously adopted to the changing standards. In addition, DSM tools should provide better support for all phases of DSL development cycle including decision, analysis, design, implementation and deployment.

In order to evaluate MS/DSL Tools and Eclipse EMF/GEF/GMF, the author has developed two Domain-Specific Language designers (one by using each tool) for modeling business entity diagrams of a banking application. Evaluation of two modeling tools was performed based on a set of criteria with help of experiences gained in development of two DSL designers. The evaluation results indicate that both DSM tools have their own strengths and weaknesses corresponding to each evaluation criteria. Advantages and disadvantages of these DSM tools are summarized as follows:

- The Ecore metamodel of EMF is more simple and robust comparing to proprietary metamodeling notation provided by MS/DSL Tools. The disadvantage of MS/DSL Tools is that it does not provide any support to OMG standards such as MOF; rather it has its own proprietary notation for metamodeling.
- Both Ecore metamodel of EMF and proprietary notation provided by MS/DSL Tools is expressive enough to visualize domain concepts of a banking application effectively.
- Regarding model transformations, Eclipse supports transformation languages (e.g. ATL) and standards (e.g. QVT) for both model-to-text and model-to-model transformations whereas MS/DSL Tools lacks an explicit technique for model-to-model transformations. Both DSM tools provide template-based approach for model-to-text transformation, which requires considerable effort to write text templates. One advantage of MS/DSL Tools is that it supports the definition of partial classes which reduces the complexity of code generation.
- Despite the fact that current versions of both DSM tools provide limited visual notations and editor functionality for generating graphical editors, DSL designers that were developed in this thesis by using MS/DSL Tools and Eclipse GMF are easy to use. However, in order to increase their maturity to the same level as UML diagram designers, tool vendors need to develop more professional editor functionalities.
- Vendor dependent commercial characteristic of DSL Tools leads to more stable versions comparing to Eclipse GMF which is more frequently upgraded to new builds.
- Important advantage of MS/DSL Tools is that it has competitive advantage due to its tight integration to Visual Studio 2005 which is one of the most popular development environments.
- Another drawback of both DSM tools is that they do not provide support for round-trip engineering which provides generation of models from existing source code. Furthermore, there is a need for better documentation support for both DSM tools.

One of the research questions was how models defined under two modeling technologies can be exchanged. It is likely that when the number of tools which provide full support for DSM increases, fragmentation between these DSM tools will also increase. Therefore, there is a need to provide bridging mechanisms between these tools to exchange models and to establish an interoperable MDD environment. In this sense, the operational bridge between MS/DSL Tools and Eclipse EMF can be built using model transformation languages like ATL. This bridge should be built at both metamodel and model level and should operate in

both directions. In this way, models conforming to domain models in MS/DSL Tools can be transformed to EMF models conforming to EMF metamodels. Once the bridge between two MDD platforms is implemented, the lesson learnt can be reused in building other exchange schemas between similar platforms, e.g. GME and EMF.

Another research questions was how DSM tools can be integrated with existing development tools. In order to integrate DSM tools to existing software development environment and tools, integration support should be realized at different levels: data/control and presentation. At data/control level integration code must be added either to the application to be developed using DSM tools or into the existing application that is to be integrated. At presentation level, a specific DSL which provides description of the mapping rules between model elements and existing applications should be defined. Besides, limitations of each DSM tool with regard to integration with existing environment should be considered.

Since Eclipse provides an extensible open source environment where a variety of tools can be easily integrated, organizations can customize modeling frameworks to better fit their DSL development process. Organizations can produce their own DSM tools with customized features like graphical modeling and model validation. Ease of extensibility of Eclipse Modeling Frameworks also facilitates integrating these frameworks to the existing development environments. On the other hand, MS/DSL Tools is tightly-coupled with Microsoft Visual Studio 2005 which makes it difficult to customize. Besides, well integration of DSL Tools to Visual Studio 2005 solves many integration concerns that Eclipse faces with.

The author is aware of the fact that one DSL like Business Entity Language (BEL) developed in this thesis is not enough to represent all domain concepts of a banking application. A typical DSL development project shall include several small DSLs rather than one big DSL which eventually will lack the quality of being concise and convenient as general-purpose modeling languages do.

Furthermore, organizations should perform their own evaluations including other commercial (e.g. MetaEdit+) and open source (e.g. GME) DSM tools to find out which tool better fits to their own DSL development project needs. When selecting DSM tools for evaluation, one important issue is to pay attention whether a DSM tool intends to address vertical domains or horizontal domains. For instance, MS/DSL Tools focus more on horizontal domains (e.g. user interface construction, data access) whereas MetaEdit+ focuses more on vertical domains (e.g. banking, health care). Furthermore, vendor-executed tool comparisons should not be taken as a basis due to subjectivity involved in these evaluations.

Main motivation behind this thesis was to provide better visibility to existing MDD tools which play a vital role for realization of software industrialization, third wave in software industry. Apart from second-wave, software process improvement, benefits of MDD can be measured more reasonably in long-term. MDD provides significant increase in productivity (due to automation of labor-intensive and error-prone development activities) as well as software quality benefits in terms of efficiency, scalability and reliability.

Finally, in order to realign software engineering curriculum in universities to the needs of software industrialization, the author recognizes the need of introduction of new courses comprised of theoretical work of key concepts, principles and standards of MDD as well as practical work using existing MDD tools.

8 FUTURE WORK

The outcome of this thesis is a set of guidelines which are comprised of advantages and disadvantages of Microsoft DSL Tools and Eclipse Modeling Frameworks. These guidelines help software practitioners to select appropriate DSM tools for their own DSL development projects. This thesis provides an evaluation of Microsoft DSL Tools and Eclipse Modeling Frameworks based on a set of evaluation criteria. The author has selected a small number of issues (shown in Table 4) to evaluate due to time restrictions of this thesis project. However, there are some other issues that could have an impact on the evaluation results as well as DSL development process. It would be interesting to evaluate those issues below in future work and to prepare guidelines for these issues regarding pros and cons of each DSM tool.

- In this thesis, the author's approach towards DSL development has been more qualitative. Further research is needed to determine how the costs and benefits of developing DSLs can be reasonably quantified. The cost calculations shall include time and resources spent for creation of models and model transformations for different DSM tools.
- Since template-based model-to-text transformation requires better support from both Eclipse and MS/DSL Tools, code generation capabilities (including both C# and Java based text templates) shall be compared using newer versions of both DSM tools. Furthermore, it would be interesting to develop a custom run tool (or a file generator) instead of using code generators provided by DSM tools (e.g. DSL Tools provides custom generator which generates C# files from text templates)
- Further work in how DSM tools support existing model-to-model transformation technologies such as OMG's QVT standard and ATL component of the GMT project shall be carried out.
- Another important area of DSM is simulation and model transformations. Further work is required regarding how DSM tools can support simulation and complex model transformations for model verification.
- Traditional base frameworks and toolsets (e.g. Visual Studio and Eclipse) are likely to evolve into a series of horizontal DSLs (e.g. Class Designer, UI DSL, Web Service DSL, Distributed Operations DSL) and there will be lots of third party vertical DSLs [69]. Further evaluations of horizontal and third party vertical DSLs shall be performed.
- Another open issue is that how knowledge engineering methodologies like CommonKADS [76] and ontology development can be used in domain analysis phase of DSL development for capturing, representing and analyzing domain knowledge.
- Productivity benefits of MDD can be measured in terms of increase in number of lines of code produced per staff months and reduction in the turnaround time. It would be interesting to measure productivity and efficiency of compilers/code generators developed using MDD tools in terms of compile time, execution time, execution size, generated code size, etc.
- It would be interesting to compare both Eclipse Modeling Frameworks and MS/DSL Tools with other DSM tools such as MetaEdit+ and GME.

9 REFERENCES

[1] J. Greenfield and K. Short, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, John Wiley and Sons, 2004.

[2] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2001.

[3] Visual Studio 2005 Team System Modeling Strategy and FAQ (May 2005), http://msdn.microsoft.com/vstudio/DSLTools/default.aspx?pull=/library/en-us/dnvs05/html/vstsmodel.asp, January 2007.

[4] K. Czarnecki and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, 2000.

[5] P. C. Smolik, Mambo Metamodeling Environment, A dissertation submitted in partial fulfillment of the requirements for the degree of doctor of philosophy, Brno University of Technology, Czech Republic, 2006.

[6] J. Bézivin and O. Gerbé, Towards a Precise Definition of the OMG/MDA Framework, In *Proceedings of the 16th IEEE international Conference on Automated Software Engineering*, pages 273-280, IEEE Computer Society, Washington, DC, 2001.

[7] J. Bezivin and E. Breton, Applying The Basic Principles of Model Engineering to The Field of Process Engineering, *The European Journal for the Informatics Professional*, V(5):27--33, 2004.

[8] J.M. Favre, Megamodelling and Etymology, In *Proceedings of Dagstuhl Seminar 05161 on Transformation Techniques in Software Engineering*, Dagstuhl, Germany, 2005.

[9] PlanetMDE, A Web Portal for The Model Driven Engineering Community, http://planetmde.org, January 2007.

[10] J. Bézivin, In Search of a Basic Principle for Model-Driven Engineering, CEPIS, UPGRADE, *The European Journal for the Informatics Professional*, V (2):21--24, April 2004.

[11] A Definition of MDA, ORMSC plenary session (August 2004), Montreal, http://www.dsic.upv.es/workshops/dsdm04/files/MDA-ormsc-040802.pdf, January 2007.

[12] A. Abouzahra, J. Bezivin, M. D. Del Fabro, and F. Jouault, A Practical Approach to Bridging Domain Specific Languages with UML profiles, In *Proceedings of the OOPSLA Workshop on Best Practices for Model Driven Software Development at OOPSLA'05*, San Diego, California, USA, 2005.

[13] OMG, UML Superstructure Specification, v2.0, http://www.omg.org/docs/formal/05-07-04.pdf, January 2007.

[14] OMG, MOF 2.0/XMI Mapping Specification, v2.1,http://www.omg.org/docs/formal/05-09-01.pdf , January 2007.

[15] OMG, MOF Core Specification, v2.0,http://www.omg.org/docs/formal/06-01-01.pdf, January 2007.

[16] OMG, CWM Specification, v1.1, http://www.omg.org/docs/formal/03-03-02.pdf, January 2007.

[17] OMG, MDA Guide v1.0.1, http://www.omg.org/docs/omg/03-06-01.pdf, January 2007.

[18] OMG Trademarks, http://www.omg.org/legal/tm_list.htm, January 2007.

[19] Microsoft Visual Studio 2005 SDK including Domain-Specific Language Tools Main Page, http://msdn.microsoft.com/vstudio/DSLTools/, January 2007.

[20] Eclipse Modeling Framework (EMF) Main Page, http://www.eclipse.org/emf/, January 2007.

[21]Eclipse Graphical Editing Framework (GEF) Main Page, http://www.eclipse.org/gef/, January 2007.

[22] Eclipse Graphical Modeling Framework (GMF) Main Page, http://www.eclipse.org/gmf/, January 2007.

[23] OMG Model Driven Architecture (MDA) Main Page, http://www.omg.org/mda/, January 2007.

[24] Microsoft Software Factories Main Page, http://www.softwarefactories.com/, January 2007.

[25] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. J. Grose, *Eclipse Modeling Framework: A Developer's Guide*, Addison Wesley Professional, 2003.

[26] J. Bézivin, G. Hillairet, F. Jouault, I. Kurtev, and W. Piers, Bridging the MS/DSL Tools and the Eclipse Modeling Framework, In *Proceedings of the International Workshop on Software Factories at OOPSLA'05*, San Diego, California, USA, 2005.

[27] A. Demir, Comparison of Model-Driven Architecture and Software Factories in the Context of Model-Driven Development, In *Proceedings of the Fourth Workshop on Model-Based Development of Computer-Based Systems and Third international Workshop on Model-Based Methodologies For Pervasive and Embedded Software (MBD-MOMPES'06)*, IEEE Computer Society, pages 75-83, Washington, DC, 2006.

[28] S. Kelly, Comparison of Eclipse EMF/GEF and MetaEdit+ for DSM, In *Proceedings of the OOPSLA & GPCE Workshop on Best Practices for Model Driven Software Development at OOPSLA'04*, 2004, http://www.softmetaware.com/oopsla2004/kelly.pdf, January 2007.

[29] S. Cook, Domain Specific Modeling and Model Driven Architecture, MDA Journal, January 2004, http://www.bptrends.com/publicationfiles/01-04%20COL%20Dom%20Spec%20Modeling%20Frankel-Cook.pdf, January 2007.

[30] MSDN, Overview of Domain-Specific Language Tools, http://msdn2.microsoft.com/en-us/library/bb126327(VS.80).aspx, January 2007.

[31] M. Fowler (August 2006), DSL Boundary,http://martinfowler.com/bliki/DslBoundary.html, January 2007.

[32] V. Pelechano, M. Albert, J. Munoz, and C. Cetina, Building Tools for Model Driven Development Comparing Microsoft DSL Tools and Eclipse Modeling Plug-ins, In *Proceedings of the 11th Conference on Software Engineering and Database (JISBD'06)*, Barcelona, Spain, 2006.

[33] Eclipse Modeling Project Main Page, http://www.eclipse.org/modeling/, January 2007.

[34] Eclipse Model-to-Model Transformation (M2M) Project Proposal Main Page, http://www.eclipse.org/proposals/m2m/, January 2007.

[35] DSM (Domain-Specific Modeling) Forum Main Page, http://www.dsmforum.org/, January 2007.

[36] M. Fowler (June 2005), Language Workbenches: The Killer-App for Domain Specific Languages?, http://martinfowler.com/articles/languageWorkbench.html, January 2007.

[37] A. Deursen, P. Klint, and J. Visser, Domain-Specific Languages: An Annotated Bibliography (February 2000), http://homepages.cwi.nl/~arie/papers/dslbib/, January 2007.

[38] A. W. B. Furtado, Sharpludus: Improving Game Development Experience through Software Factories and Domain-Specific Languages, Master Degree Dissertation, UFPE Informatics Center, Brazil, 2006, http://www.cin.ufpe.br/~awbf/files/AFurtadoMSc_SharpLudus.pdf, January 2007.

[39] E. G. Sirer and B. N. Bershad, Using Production Grammars in Software Testing, In *Proceedings of the 2nd Conference on Domain-Specific Languages*, pages 1-13, ACM Press, New York, NY, 1999.

[40] J. Warmer and A. Kleppe, Building a Flexible Software Factory Using Partial Domain Specific Models, In *Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling (DSM'06)*, pages 15-22, 2006, http://www.dsmforum.org/events/DSM06/Papers/2-kleppe.pdf, January 2007.

[41]Model-Integrated Computing (MIC) Main Page, http://www.isis.vanderbilt.edu/research/research.html, January 2007.

[42] Omondo EclipseUML Main Page, http://www.omondo.com/, January 2007.

[43] Compuware: Landmark Study Reveals 35 per cent Productivity Gains for Businesses Using Model-Driven Architecture, http://www.compuware.co.uk/pressroom/news/21072003_02.htm, January 2007.

[44] MetaCASE, MetaEdit+ Main Page, http://www.metacase.com/mep/, January 2007.

[45] Institute for Software Integrated Systems, Generic Modeling Environment (GME) Main Page, http://www.isis.vanderbilt.edu/projects/gme/, January 2007.

[46] X. Ke and K. Sierszecki, Generative Programming for a Component-based Framework of Distributed Embedded Systems, In *Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling (DSM'06)*, pages 113-122, 2006, http://www.dsmforum.org/events/DSM06/Papers/12-Sierszecki.pdf, January 2007.

[47] H. Krahn, B. Rumpe, and S. Völkel, Roles in Software Development using Domain Specific Modeling Languages, In *Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling (DSM'06)*, pages 150-158, 2006, http://www.dsmforum.org/events/DSM06/Papers/15-Voelkel.pdf, January 2007.

[48] OMG, Meta Object Facility (MOF) Main Page, http://www.omg.org/mof, January 2007.

[49] M. Feilkas, How to represent Models, Languages and Transformations?, In *Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling (DSM'06)*, pages 169-176, 2006, http://www.dsmforum.org/events/DSM06/Papers/17-Feikas.pdf , January 2007.

[50] H. Jonkers, M. Stroucken, and R. Vdovjak, Bootstrapping Domain-Specific Model-Driven Software Development within Philips, In *Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling (DSM'06)*, pages 204-213, 2006, http://www.dsmforum.org/events/DSM06/Papers/21-Vdovjak.pdf, January 2007.

[51] B. Selic, Model-Driven Development: Its Essence and Opportunities, In *Proceedings of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06)*, pages 313-319, IEEE Computer Society, Washington, DC, 2006.

[52] R. B. France, S. Ghosh, T. Dinh-Trong, A. Solberg, Model-Driven Development using UML 2.0-Promises and Pitfalls, *Computer*, 39(2): 59--66, 2006.

[53] OMG, UML Profiles, http://www.uml.org/#UMLProfiles, January 2007.

[54] D.C. Schmidt, Model-Driven Engineering , *Computer*, 39(2): 25--31, 2006.

[55] K. Balasubramanian, A. Gokhale, G. Karsai, J. Sztipanovits, and S. Neema, Developing Applications Using Model-Driven design Environments, *Computer*, 39(2): 33--40, 2006.

[56] B. Selic, “The Pragmatics of Model-Driven Development”, *IEEE Software*, 20(5): 19-25, 2003.

[57] T. Weigert and F. Weil, Practical Experiences in Using Model-Driven Engineering to Develop Trustworthy Computing Systems, In *Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC`06)*, pages 208-217, 2006.

[58] B. Trask and A. Roman, Leveraging Model Driven Engineering in Software Product Lines, In *Proceedings of the Tenth International Software Product Line Conference (SPLC`06)*, page 221, 2006.

[59] Eclipse.org, GMF Tutorial, http://wiki.eclipse.org/index.php/GMF_Tutorial, January 2007.

[60] M. Völter (October 2006), A Categorization of DSLs, http://voelterblog.blogspot.com/2006/10/categorization-of-dsls.html, January 2007.

[61] Build a Domain Specific Language Designer Using the DSL Tools for Microsoft Visual Studio 2005, http://download.microsoft.com/documents/australia/teched2005/hol/HOL004.pdf, January 2007.

[62] N.A. Allen, C.A. Shaffer, and L.T. Watson, Building Modeling Tools that Support Verification, Validation, and Testing for the Domain Expert, In *Proceedings of the 37th Conference on Winter Simulation*, pages 419-426, 2005.

[63] S. Kelly (February 2006), How tightly should you integrate your DSM tool and IDE, http://www.metacase.com/blogs/stevek/blogView?showComments=true&entry=3318510134, January 2007.

[64] D.S. Kolovos, R.F. Paige, T.P. Kelly, and F.A.C. Polack, Requirements for Domain-Specific Languages, In *Proceedings of the First ECOOP Workshop on Domain-Specific Program Development*, co-located with ECOOP’06, Nantes, France, 2006.

[65] D. Wile, Lessons Learned from Real DSL Experiments, In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, 2003.

[66] M. Mernik, J. Heering, and A. M. Sloane, When and How to Develop Domain-Specific Languages, *ACM Computing Surveys*, 37(4):316–344, 2005.

[67] N.H. Christensen, Domain-Specific Languages in Software Development and the relation to partial evaluation, PhD thesis, DIKU, Dept. of Computer Science, University of Copenhagen, Denmark, July 2003.

[68] M. Völter, T. Stahl, J. Bettin, A. Haase, S. Helsen, and K. Czarnecki, *Model-Driven Software Development: Technology, Engineering, Management*, John Wiley and Sons, 2006.

[69] E. O'Tuathail, Domain-Specific Languages, http://www.clipcode.biz/workshops/SPL_SoftwareFactories_GAT_DSLs.pdf, January 2007.

[70] W. S. Humphrey, *Managing the Software Process*, Addison-Wesley, 1989.

[71] S. Zahran, *Software Process Improvement - Practical guidelines for Business success*, Addison-Wesley, 1997.

[72] The Eclipse Modeling Framework Technology project, http://www.eclipse.org/emft/projects/, January 2007.

[73] MSDN, Validation Overview for Domain-Specific Languages, http://msdn2.microsoft.com/en-us/library/bb126419(VS.80).aspx, January 2007.

[74] J.C. Gundy, J.G. Hosking, N. Zhu, and N. Liu, Generating Domain-Specific Visual Language Editors from High-level Tool Specifications, In *Proceedings of the 2006 IEEE/ACM International Conference on Automated Software Engineering*, Tokyo, IEEE, September 2006.

[75] 6th OOPSLA Workshop on Domain-Specific Modeling (OOPSLA06) Preface, http://www.dsmforum.org/events/DSM06/Papers/preface.pdf, January 2007.

[76] CommonKADS, http://www.commonkads.uva.nl/frameset-commonkads.html, January 2007.

[77] F. Jouault and J. Bézivin, KM3: a DSL for Metamodel Specification, In *Proceedings of 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems*, LNCS 4037, Bologna, Italy, pages 171-185, 2006, http://www.sciences.univ-nantes.fr/lina/atl/www/papers/KM3-FMOODS06.pdf, January 2007.

[78] K. Pauls (2004), RECOMMEND - Round-trip Engineering for COMponent-based Model-drivEN Development, Freie Universitat Berlin, Fachbereich Mathematik und Informatik, Germany, http://www.inf.fu-berlin.de/inst/ag-ti/tr-b-04-15.ps, January 2007.