Modern Software Engineering Concepts and Practices:
Advanced Approaches

Ali H. Doğru
Middle East Technical University, Turkey

Veli Biçer
FZI Research Center for Information Technology, Germany

Information Science REFERENCE

INFORMATION SCIENCE REFERENCE
Hershey · New York

Senior Editorial Director: Kristin Klinger
Director of Book Publications: Julia Mosemann
Editorial Director: Lindsay Johnston
Acquisitions Editor: Erika Carter
Development Editor: Joel Gamon
Production Coordinator: Jamie Snavely
Typesetters: Keith Glazewski & Natalie Pronio
Cover Design: Nick Newcomer

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Copyright © 2011 by IGI Global. All rights reserved. No part of this publication may be reproduced, stored or distributed in any form or by any means, electronic or mechanical, including photocopying, without written permission from the publisher. Product or company names used in this set are for identification purposes only. Inclusion of the names of the products or companies does not indicate a claim of ownership by IGI Global of the trademark or registered trademark.

Library of Congress Cataloging-in-Publication Data

Modern software engineering concepts and practices : advanced approaches / Ali H. Doğru and Veli Biçer, editors.
p. cm.
Includes bibliographical references and index.
Summary: "This book provides emerging theoretical approaches and their practices and includes case studies and real-world practices within a range of advanced approaches to reflect various perspectives in the discipline"-- Provided by publisher.
ISBN 978-1-60960-215-4 (hardcover) -- ISBN 978-1-60960-217-8 (ebook) 1. Software engineering. I. Doğru, Ali H., 1957- II. Biçer, Veli, 1980-
QA76.758.M62 2011
005.1--dc22
2010051808

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

All work contributed to this book is new, previously-unpublished material. The views expressed in this book are those of the authors, but not necessarily of the publisher.

Chapter 14
A Software Cost Model to Assess Productivity Impact of a Model-Driven Technique in Developing Domain-Specific Design Tools

Achilleas Achilleos
University of Cyprus, Cyprus

Nektarios Georgalas
British Telecom (BT) Innovate, UK

Kun Yang
University of Essex, UK

George A. Papadopoulos
University of Cyprus, Cyprus

ABSTRACT

Programming languages have evolved through the course of research from machine dependent to high-level "platform-independent" languages. This shift towards abstraction aims to reduce the effort and time required by developers to create software services. It is also a strong indicator of reduced development costs and a direct measure of a positive impact on software productivity. Current trends in software engineering attempt to raise further the abstraction level by introducing modelling languages as the key components of the development process. In particular, modelling languages support the design of software services in the form of domain models. These models become the main development artefacts, which are then transformed using code generators to the required implementation. The major predicament with model-driven techniques is the complexity imposed when manually developing the domain-specific design tools used to define models. Another issue is the difficulty faced in integrating these design tools with

DOI: 10.4018/978-1-60960-215-4.ch014

Copyright © 2011, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

model validation tools and code generators. In this chapter a model-driven technique and its supporting model-driven environment are presented, both of which are imperative in automating the development of design tools and achieving tools integration to improve software productivity. A formal parametric model is also proposed that allows evaluating the productivity impact in generating and rapidly integrating design tools. The evaluation is performed on the basis of a prototype domain-specific design tool.

INTRODUCTION

The escalating and rapidly changing user requirements contribute towards increased complexity in the software development process. Furthermore, the advancements and diversity in technologies currently present escalate further the complexity introduced to the process. Consequently, the software engineering community seeks innovative and abstract techniques that provide the capability to scale down the complexity problem, in order to simplify and expedite the development of domain-specific software services. The objective is to provide "platform-independent" techniques that support the creation of software services at an abstract level steering the developer away from platform-specific implementation complexities.

During the early years of *Software Engineering* the difficulties and pitfalls of designing complex software services were identified and a quest for improved software development methodologies and tools began (Wirth, 2008). The first steps towards this goal introduced formal notations, known as programming languages, used mainly for performing mathematical analysis computing tasks. Examples of such numerical programming languages are *FORTRAN, Algol and COBOL*. Since then demand for more powerful software applications that perform complex computational tasks, rather than simple mathematical tasks, has largely grown. Therefore, it was acknowledged that more competent programming languages, software tools and automation capabilities were required to successfully implement these complex computing tasks (Wirth, 2008).

The software engineering discipline concentrated on the development of high-level programming languages, which simplify the development of software applications. A minor setback in the inclination towards programming abstraction was the machine dependent *C language*. As Wirth (2008, p. 33) clearly states:

"From the point of view of software engineering, the rapid spread of C therefore represented a great leap backward..... It revealed that the community at large had hardly grasped the true meaning of the term "high-level language", which became a poorly understood buzzword. What, if anything, was to be "high level" now?"

Although the *C language* provides efficiency in creating simple hardware-dependent software services, it proved scarce and complex in developing, testing and maintaining large and versatile software applications (Wirth, 2008). The lessons learned from using the *C language* guided though software engineers to devise abstract and disciplined software techniques, like the predominant *Object-Oriented (OO)* programming model (Chonacky, 2009). On the basis of this model different 3GLs were developed such as Smalltalk, C++, Java and C#. These languages aimed to raise the level abstraction in software engineering and facilitate the definition of disciplined, systematic and object- oriented techniques for software development. 3GLs allow building advanced software

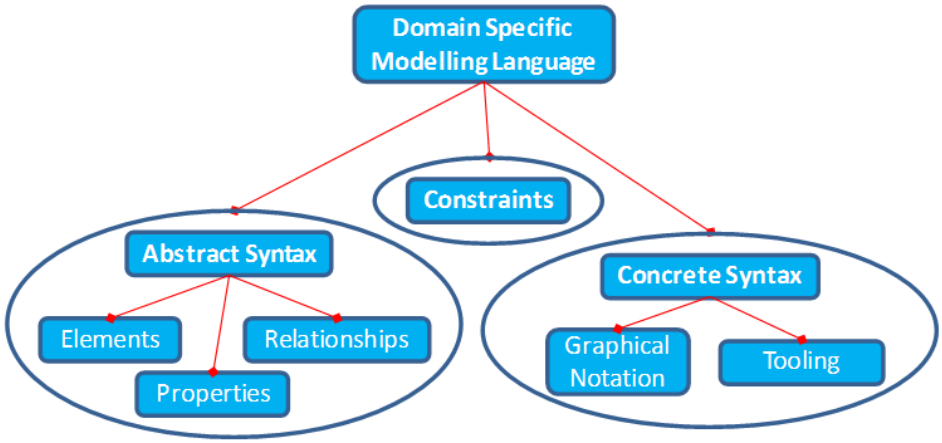
services that feature visual objects (e.g. buttons, labels) with distinct state and behaviour.

The continuous development of programming languages can be considered as a sign of healthy evolution (Chonacky, 2009), which stems from the necessity to overcome complexities imposed by the software development process. In particular, this pragmatic progress leaded to the creation of many Domain-specific Languages (DSLs) that tackle software development at a higher abstraction level (Deursen et. al., 2000; Graff et. al., 2007; Iscoe et. al., 1991) and introduce a shift from code-centric to model-centric development (Staron, 2006; Afonso et. al., 2006). This category of languages can be divided into two closely related subcategories: (i) text-based DSLs and (ii) model-based DSLs. Examples of such DSLs are *Matlab, Simulink* and *SolidWorks*, which describe and/or combine text-based and modelling software capabilities. These languages are proven to be highly competent in terms of their targeted problem domain rather than being all-around General-Purpose Languages (GPLs). Therefore, the semantics of these languages can be interpreted precisely to a platform-specific implementation since they are very precise and leave no room for miscellaneous interpretations (Evermann et. al., 2005; Clark et. al., 2004). The well-acknowledged success of DSLs comes as an outcome of the following: (i) satisfying the domain's requirements, (ii) using proficient software tools to support them and (iii) restricting user input to properties of the target domain while providing easy access to artefacts (Sprinkle et. al., 2009). Moreover, they provide modelling and coding simplicity and aim for platform-independence (Chonacky, 2009).

Domain-specific Modelling (DSM) refers to the activity that allows developing and using graphical DSLs. It is a software engineering paradigm that raises the level of abstraction by introducing models as the prime entities of the development process. Although DSM is currently at its peeks, it is rather a revived and improved concept that shifts the focus to narrower application domains of increased abstraction (Sprinkle et. al., 2009). In particular, early programming languages such as FORTRAN and COBOL can be also regarded as DSLs, which embrace though the much broader domains of scientific and business computing. As aforesaid the added-value of DSLs lies in their focused expressive power and not their broad applicability (Freudenthal, 2009). Therefore, the success of DSLs lies in addressing smaller domains and defining concepts restricted to these problem-specific domains. In addition, tools have evolved significantly in terms of providing the software capabilities that allow defining DSLs, validating and transforming models and automatically generating the implementation from models.

In this chapter we introduce a model-driven technique and a supporting environment, which allow automatically generating concrete, customisable, extensible and bug-free domain-specific design tools. Our focus is to provide a quantitative evaluation method that considers a large number of parameters to assess the impact of the proposed model-driven technique and its supporting environment on software productivity. In particular, the evaluation method should provide the capability to assess the productivity impact in generating and rapidly integrating design tools into a unified environment. The evaluation is based on a well-documented and widely accepted formal model (i.e. COCOMO II.2000 - Post-Architecture model), which allows estimating the effort, time and cost related to software development (Boehm et. al., 2000; Chen et. al., 2005). In particular, due to the nature of the model-driven technique the evaluation method takes into consideration an extension of the Use of Software Tools (TOOLS) parameter defined in the model. Using this extension the critical role of software tools is heavily considered in the estimation of the impact on software productivity. Finally, the evaluation takes into consideration the following requirements, which should be satisfied to efficiently accomplish design tools generation. Figure 1 illustrates

Figure 1. The necessary artefacts for defining a domain-specific modelling language



explicitly these requirements (i.e. artefacts), which are imperative for developing a DSL and its supporting design tool.

R1. *A standardised language is required that provides rich syntax, semantics and a supporting tool for defining the abstract syntax of DSLs.*

R2. *The capability to define constraints should be provided using a software tool that conforms to a standardised language and allows defining rules that restrict the abstract syntax of the DSL.*

R3. *A widely-used modelling language and a supporting tool is required that allows defining the concrete syntax of DSLs.*

R4. *The capability to merge the abstract syntax, constraints and concrete syntax into a common representation (i.e. a model) that allows generating automatically the modelling tools of DSLs.*

The chapter is structured as follows: Section 2 presents background information on Model-Driven Development (MDD) environments, which target explicitly the generation of domain-specific design tools. Moreover, Section 3 introduces related work that uses MDD environments for automating the development of domain-specific design tools. In Section 4 we present the model-driven technique with particular focus in *automating the generation of DSLs and their supporting modelling tools*. Section 5 presents the *architectural design of the proposed model-driven environment*. Following, Section 6 showcases the automatic generation of a prototype design tool used in the *Product Lifecycle Management process*. A quantitative evaluation is then performed on the basis of the above requirements and the selected software cost estimation model. Finally, Section 7 summarises, concludes and proposes directions for future research work.

BACKGROUND

The progress of research work on MDD acknowledges that practising domain-specific modelling in conjunction with the Model Driven Architecture (MDA) paradigm (Frankel, 2003; Kleppe, 2005; OMG MDA, 2003) can increase software productivity (Kelly & Pohjonen, 2009; Balasubramanian et. al., 2005). These research efforts recognize also the main issue with domain-specific modelling, which is the necessity to rapidly develop the modelling tools that support the DSLs. The growth

of MDD environments and the capabilities they currently provide allow overcoming this issue to a great extend. Most of these environments provide automation in developing domain-specific languages and their supporting modelling tools. However, deficiencies still exist due to the failure to adopt a common, systematic model-driven technique and align fully with the MDA standards. In this section we present the most competent and widely-used environments, which are capable of providing proprietary or standardised support to the proposed model-driven technique, to identify possible limitations.

The Generic Modelling Environment (GME) is a research environment that practises Model Integrated Computing (MIC). MIC is actually a methodology developed to steer the GME in the development of embedded software systems. The tool stemmed from earlier research on domain-specific visual programming environments to become a highly competent domain-specific modelling environment (Molnár et. al., 2007). In particular, it can be adapted and configured at the meta-level to obtain a domain-specific modelling tool that is tailored to an explicit engineering domain. The GME defines a proprietary metamodelling language that includes the concepts built-in to the tool. Therefore, a DSL can be defined using a UML-like Class Diagram (i.e. metamodel) that describes the concepts of the engineering domain. Furthermore, it provides additional tools for defining domain rules using the Object Constraint Language (OCL) (OMG OCL, 2005) and GME-specific configurable model visualization properties. Although MetaGME is conceptually similar to the Meta-Object Facility (MOF) specification (OMG MOF, 2005) it is still not MOF-based. Hence, model-to-model transformations need to be defined to translate between the two languages (Emerson & Sztipanovits, 2004). Essentially, the requirement for compliance to MDA standards and the common interest on metamodelling motivated the GME research community to bridge with the Eclipse modelling community into a joined initiative.

AndroMDA is an extensible generator environment that utilises UML tools to define models that can be transformed to a platform-specific implementation. In particular, the environment adheres to the MDA paradigm by utilising UML profiling rather than focusing on metamodelling. The environment is bound mainly to the notion of a "cartridge", which allows processing model elements with specific stereotypes using the template files defined within the cartridge. Templates describe how the models are transformed to deployable components that target well-known platforms such as J2EE, Spring,. NET. Consequently, the environment does not provide any inherent support for metamodelling and domain-specific modelling, since it is largely based on UML. In a latest snapshot release (i.e. AndroMDA 4.0-M1) the environment shifts its focus towards metamodelling using Eclipse-based modelling implementations and the concept of domain-specific modelling.

The XMF-Mosaic is a model-driven environment, which is based on the concept of metamodelling and provides support for domain-specific modelling. In particular, the metamodelling environment provides advanced capabilities for defining and generating DSLs and their supporting modelling tools. Furthermore, the software tools provided by the model-driven environment are largely aligned with the MDA specifications defined by the Object Management Group (OMG). Although the XMF-Mosaic is a powerful open-source model-driven environment built on top of the Eclipse platform, its development was terminated. In its latest version the tool interoperates closely with the Eclipse modelling implementations. This is basically due to the wide-acceptance of these implementations by the larger modelling community. Finally, the environment is to become part of the Eclipse Generative Modelling Technologies (GMT) project, which sole purpose is to

produce a set of prototypes in the area of Model Driven Engineering.

Microsoft DSL Tools is a powerful model-driven environment that supports model-driven development with particular focus on domain-specific modelling. The software factory comprises a bundle of proprietary software tools developed on top of the Visual Studio development platform. In particular, DSL Tools facilitate explicitly the definition of the abstract syntax and the constraints that govern the DSL, which provides the capability to validate the designed models. Furthermore, the capability is provided to define the concrete syntax of the modelling language, in order to facilitate the generation of the required modelling tools for the language. The only predicament with the DSL factory is the necessity to learn how to use the proprietary languages and tools since the factory does not conform to the OMG specifications. Microsoft Corporation recently joined the OMG in an attempt to meet the standards so as to fulfil their strategy and assist in taking modelling into mainstream industry use.

Borland Together is the final model-driven environment examined in this chapter that provides the necessary tools to support the definition of DSLs and the generation of the accompanying modelling tools. First, the environment allows defining the abstract syntax and constraints that govern domain models. Moreover, the concrete syntax can be defined to provide a graphical notation for the artefacts of the DSL and the necessary tooling for the generated modelling tool. The environment is composed mainly by open-source Eclipse modelling implementations, which are customised to improve user experience and aid designers and developers to perform efficiently the required modelling and implementation tasks. The Eclipse implementations composing the environment are highly compliant to the OMG standards and are widespread and widely-known to an extensive group of designers and developers. Borland Together, like Microsoft DSL tools, is a commercial product that is not freely available and as a result does not allow designers and developers to extend it or customise it to satisfy their explicit requirements.

Most of these Eclipse implementations were introduced as new software capabilities in Borland Together 2008. These implementations are equivalent to the ones composing the model-driven environment initially proposed in (Achilleos et. al., 2007) and evaluated in this chapter. To the author's best knowledge when the environment was initially designed the existing literature and documentation (Borland, 2006) did not disclose such software capabilities. This does not abolish the fact that analogous attempts were made by Borland during that period to develop and deliver a unified model-driven environment with analogous software capabilities. Regardless of that fact, the objective of this chapter is not to perform a comparison of existing model-driven environments but rather to propose an evaluation method that can be applied for each environment to assess their impact on software productivity. In particular, the objective is to evaluate the capability of the environment to support a model-driven technique for automatically generating domain-specific modelling languages.

RELATED WORK

Different MDA approaches have been proposed in the literature that attempt to automate the development of DSLs, so as to simplify MDD. An approach that differentiates from mainstream DSL development (Santos et. al., 2008) proposes the extension of generic frameworks with an additional layer that encodes a DSL. The approach is solely based on a generic language workbench that allows extracting DSL concepts (i.e. DSL metamodel) from the DSM layer and transforming model instances into code that conforms to that particular DSM layer. Thus, developers are able to define DSL models like if they were using a conventional modelling tool. Moreover, the

generic language workbench allows processing domain models for generating code, rather than developing individual code generators for each DSL. The main shortcoming is that the definition of a concrete syntax for the DSL is not addressed by the approach but it is regarded as a separate issue that is handled independently from the abstract syntax. We argue though that the definition of a DSL should involve also the specification of its concrete syntax.

As aforesaid, GME is a metamodelling environment that enables the creation of domain-specific modelling environments from metamodels (Lédeczi et. al, 2001). It uses the MetaGME metamodelling language that allows defining domain concepts in a proprietary form, which is similar to a UML class diagram. Consequently, since the metamodel is proprietary, it can only be used within the GME environment and cannot be imported in different modelling tools; e.g. UML tools. This limits the applicability of the domain-specific modelling language to designers and developers that are acquainted with GME. In addition, designers and developers are not familiar with the domain concepts described in such a proprietary metamodel and cannot comprehend and transform as a result the domain models. Finally, the flexibility of DSL definition is restricted to the semantics of MetaGME and does not conform to a widely used metamodelling language (e.g. MOF) that provides a richer set of semantics.

A comparable approach (Zbib et. al., 2006), which follows the conventional DSL development process proposes the automatic generation of domain-specific modelling editors directly from metamodels. In particular, the metamodel is defined as an extension of the UML metamodel that captures domain modelling concepts. This can be described as the notion of UML profiling where each stereotype of the DSL extends an artefact of the UML metamodel; e.g. class, package, attribute. The benefit of using such an approach is that the metamodel can be imported and used in many UML tools. However, no standard way is defined to access model stereotypes in these UML tools, so as to enforce constraints and develop the necessary code generators. In addition, as admitted also in (Zbib et. al., 2006), there is greater flexibility in defining the DSL using MOF constructs; rather than being bounded by the UML semantics. Hence, we argue in this work that an approach that adheres to the MOF specification (i.e. EMF) and utilises an open-source MDD environment is largely beneficial and preferred. Furthermore, this work proposes an evaluation method that allows determining the efficiency and applicability of the MDA approach. This is an important point that is not addressed by existing work.

AUTOMATING THE DEVELOPMENT OF DOMAIN-SPECIFIC MODELLING LANGUAGES

As aforementioned, the principal issue that hinders the application of MDD is the difficulty faced with the development of domain-specific modelling languages (DSMLs). Note that we refer to the development of a DSML, rather than its definition, since it involves both the definition of the DSML and the implementation of its necessary supporting modelling tool. In particular, each DSML requires a supporting modelling tool that allows designing models that conform to the syntax, semantics and constraints of the DSML. Developing a DSML from scratch involves a time-consuming and error-prone process that necessitates high development effort; especially the implementation of the modelling tool (Nytun et. al., 2006). Consequently, the following questions arise that necessitate effective solutions for rapidly developing a DSML:

i. How to define the abstract syntax and constraints of the modelling language?
ii. How to define the concrete syntax of the modelling language?

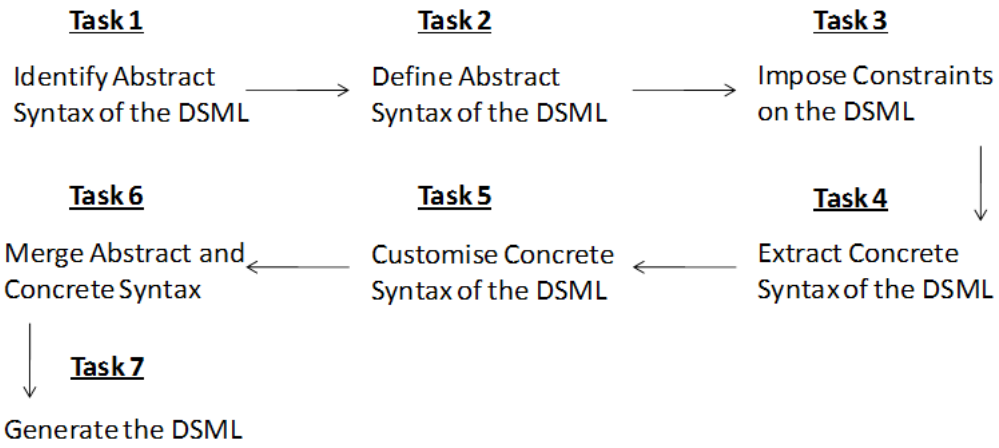
iii. How to develop a supporting software modelling tool for the language?

In this chapter we argue that explicit focus should be given to software tools in order to improve automation in domain-specific modelling tools generation. In particular, the capabilities of model-driven development tools should be fully exploited to automate the definition of DSMLs and the generation of offspring domain-specific design tools. The idea put-forward in this chapter is to utilise common, standardised and widely-used specifications to automate the development of DSMLs. Therefore, since existing MDA specifications do not provide the necessary tooling, we need to identify and/or develop software tools with high conformance to the standards. Furthermore, a disciplined and systematic model-driven technique is required that automates the development of DSMLs by utilising the capabilities of the selected software tools.

Figure 2 presents such a model-driven technique that refers to the primary phase of the methodology introduced by Achilleos et. al. (2008). This technique illustrates the tasks undertaken to accomplish the generation of DSMLs. Irrespective of the model-driven environment used, these tasks should form the baseline in order to effectively achieve increased automation in DSMLs generation. The primary task involves a requirements analysis, which helps to identify domain concepts and formulate the *Abstract Syntax* of the modelling language. In particular, the elements, properties and relationships are identified that symbolize the concepts of the domain. These concepts are then represented using a graphical notation that defines the *Abstract Syntax* of the modelling language.

The next task involves restricting the design of models to non-erroneous instances by imposing the necessary rules onto the *Abstract Syntax* of the language. This enables the execution of the third task because it allows extracting the *Concrete Syntax* of the language from its *Abstract Syntax* using model-to-model transformations. The *Concrete Syntax* of the language maps the language's domain concepts to a suitable graphical representation. For instance, an element of the language maybe mapped to a rectangle figure while a property of the language maybe mapped to a label figure. Furthermore, Task 5 illustrates the capability to customise the graphical representation of the language for human structuring purposes; i.e. improve understanding of the designed models. The next task involves merging the *Abstract* and *Concrete Syntax* of the language into a common representation that includes all the required arte-

Figure 2. Model-driven technique for automating DSMLs generation



facts of the modelling language to facilitate its tool generation (i.e. Task 7). The execution of the final task is based on the capability to translate the common representation of the modelling language to the required implementation using an existing code generator. The resulting code implements a domain-specific modelling tool that conforms to the abstract syntax, constraints and concrete syntax of the defined modelling language.

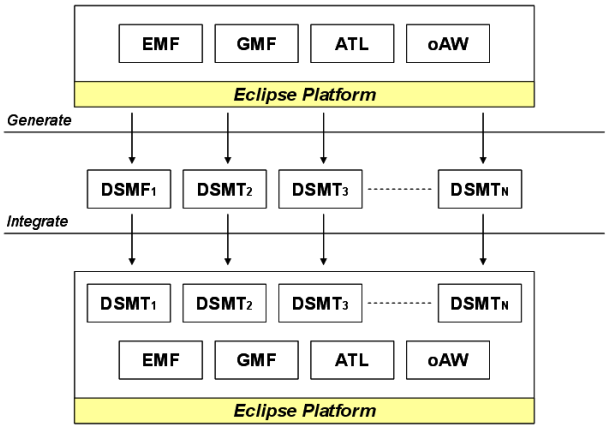
The technique provides a set of unambiguous tasks that steer the development of DSMLs. In addition the nature of the tasks allows using model-driven software tools that provide the capability to support and automate their execution. The next section describes an architectural design and proposes an environment composed by a set of Eclipse modelling implementations to support and automate the development of DSMLs.

ARCHITECTURAL DESIGN OF THE MODEL-DRIVEN ENVIRONMENT

Architectural design refers to the composition of the necessary components of a system into a coherent unit that follows a methodology for accomplishing explicit tasks in an efficient manner. The architectural design described in this subsection is based on the plug-in architecture of the Eclipse platform. Eclipse is a software platform designed for building Integrated Development Environments (IDEs) and arbitrary tools (IBMC, 2009). Hence, in accordance to the Eclipse architecture each developed software tool can be installed directly as a plug-in of the platform. The only requirement is to export the deployable plug-in (i.e. a packaged JAR file) into the "plugins" directory of the Eclipse platform. This is a dedicated directory for loading software tools or capabilities during start-up, which can be used as necessary by the designer or developer. Keeping in line with the architecture of the Eclipse platform allows satisfying the main prerequisite, which refers to the automatic generation and rapid deployment of domain-specific design tools. Furthermore, the Eclipse platform provides an extensive library of software tools many of which are dedicated to modelling and adhere to the MDA specifications.

In principle the architectural design of the environment comprises of core software tools, which support the generation of offspring domain-specific design tools. The generated design tools can be integrated directly into the model-driven environment to compose a domain-specific software service creation environment. Figure 3 illustrates the architectural design of the environment; composed by four core modelling components (i.e. software tools) developed by the Eclipse

Figure 3. Architectural design of the model-driven environment



modelling community and associated alliances. Note that the selection of these four components is not an arbitrary one but it is decided on the basis of the requirements proposed and examined by Achilleos et. al. (2007). The rationale behind the components' selection can be summarised into three key points: (i) the components should provide the necessary software capabilities to support the generation of domain-specific modelling tools (DSMTs), (ii) the components should conform to the MDA standards and (iii) the components should provide the required capabilities for transforming and generating code from models. Figure 1 illustrates the components that are namely, the Eclipse Modelling Framework (EMF), Graphical Modelling Framework (GMF), Atlas Transformation Language (ATL) and openArchitectureWare (oAW).

The root component is the EMF that started initially as an implementation of the Meta-Object Facility formal specification (OMG MOF, 2005). Both describe (meta-) modelling languages that facilitate the definition of domain-specific modelling languages. As a matter of fact they are conceptually similar and express comparable metamodelling concepts (Gerber & Raymond, 2003; Mohamed et. al., 2007). In principle EMF emphasises on the development of the essential tooling for defining metamodelling concepts, while the MOF specification provides more rigorous and expressive meta-modelling concepts for defining modelling languages; i.e. metamodels. In its current version, that is MOF 2.0, the OMG introduces a subset of the concepts described in the full specification, called Essential MOF (EMOF). The EMOF metamodelling language is conceptually identical to EMF, whereas differences are predominantly on naming. Consequently, EMF can read and write serialisations of the EMOF metamodel. As it is realised EMF has influenced heavily the MOF specification towards the critical direction of software tools integration and can be considered in this aspect as the most suitable candidate to drive the vision of model-driven development.

As aforesaid, the EMF is the heart of the environment that allows defining DSMLs using its *Ecore metamodelling language*. In particular, it allows defining the abstract syntax and semantics of the modelling language in the form of a domain metamodel. Furthermore, it provides a code generation capability that is based on Java Emitter Templates (JET) engine. This software capability enables the transformation of the metamodel into EMF-based Java implementation code, which is delivered as deployable plug-ins. The model plug-in provides the Java interfaces and implementation classes that represent the artefacts of the modelling language and the adapter plug-in provides the implementation classes that adapt the domain metamodel classes for editing and display. The final generated editor plug-in provides the classes that implement a modelling editor that conforms to the tree-based representation of the EMF. This editor supports the definition of domain models that conform to the modelling language in the form of abstract trees that include parent nodes and children as leafs.

The GMF is another important component of the environment that complements the functionality of the EMF. A modelling language requires apart from its abstract syntax and a concrete syntax that defines the graphical notation and the palette of a visual modelling tool. This is where the GMF comes in place since it provides the necessary software capabilities that allow deriving the concrete syntax of the modelling language from its abstract syntax. The concrete syntax of the modelling language is defined, in accordance to the terminology of the GMF, using the graphical and tooling metamodels. The former describes the graphical notation (e.g. rectangles, ellipses, arrows) that map to the abstract concepts defined in the Ecore metamodel, while the latter describes the tooling capabilities of the modelling editor, which are basically the palette buttons that enable its drag-and-drop functionality.

Having at hand the domain, graphical and tooling metamodels we can combine them using additional software capabilities of the GMF into a mapping metamodel to generate the visual editor plug-in. The plug-in includes the implementation classes that contribute the functionality of a structured GMF-based editor. Therefore, the set of generated plug-ins composes a fully-fledged domain-specific modelling tool, which is integrated into the original environment to deliver a software service development environment. Note that, a problem domain can be described by a single or multiple complementary modelling languages. Hence, multiple design tools might be generated and integrated into a unified environment for software service development; as illustrated in Figure 3. Examples of our work reveal that dividing the problem domain into smaller complementary sub-domains aids in terms of reducing models complexity and improve understanding (Achilleos et. al., 2008, Georgalas et. al., 2007). Finally, apart from the components that deal with the development of DSLs, the environment comprises of two supplementary frameworks that aid the transformation of models and the generation of implementation code from domain models. In this chapter, the focus is basically on the automation of the development of DSMLs and their accompanying tools. Consequently, it is out of the scope of this chapter to provide details on the operation of these frameworks.

A PROTOTYPE DESIGN TOOL FOR PRODUCT LIFECYCLE MANAGEMENT

The rapid development of large volumes of industrial software products and services is generally based on automated Product Lifecycle Management (PLM) systems (Georgalas et. al., 2009). This type of systems merge together all engineering disciplines involved and aid organisations to manage the complexity of the software development process. Telecommunication providers have recently began adopting such systems (i.e. PLM systems) because technologies such as 3G and IP are currently common practice also in the communications field. Furthermore, companies that are not inherently associated to the telecommunications field have entered the market and competition became incredibly fierce. Another factor that contributed in the adoption of PLM systems is the complexity involved in developing new software products and services. Mainly the requirement to assemble diverse components and services developed by different vendors introduces immense complexity that needs to be effectively managed. Therefore, telecommunication providers decided to adopt and adapt the PLM process, whose success is acknowledged in other industrial fields, so as to expedite and increase the efficiency in developing, deploying and offering software product and services (Georgalas et. al., 2009).

Developing a Product Lifecycle Management Design Tool

This subsection presents an industrial-based case study that involves the development of a prototype domain-specific design tool. The developed and adopted product design tool allows designers to unambiguously model products, share product specifications with other stakeholders and exchange product data amongst different Operational/ Business Support Systems (OSS/BSS) in different formats. The objective is to tackle the deficiencies introduced to the PLM process by the current techniques and tooling, used to develop software products and services. In particular, Georgalas et. al. (2009) identify the following issues with the PLM process:

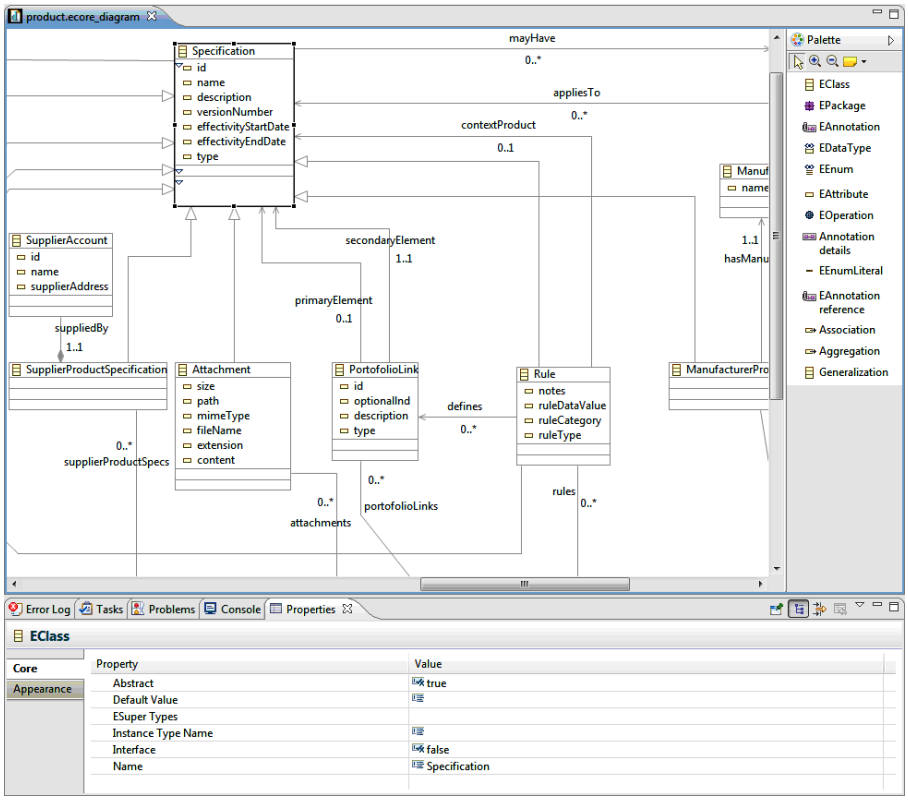
1. Current practice does not automatically drive the process from the formulation of the concept all the way through to the deployment of the product in the OSS.

2. It does not minimize the effort spend by the iterative interactions amongst the managers, designers and developers involved.
3. It does not provide and maintain an enterprise-wide understanding of the software product, mainly due to the method high-level product information is disseminated; i.e. enormous MS Word documents.

In this chapter we utilize the proposed model-driven technique and the accompanying environment to develop a domain-specific design tool that steers efficiently the PLM process. It should be noted that in this PLM case study we have used both the proposed environment and Borland Together 2008 to perform a preliminary comparison during the evaluation phase. The design tool is based on the abstract syntax, constraints and concrete syntax of the product modelling language used to generate it. The language is actually derived from a corresponding information model that defines the necessary concepts, which allow a designer to specify information regarding a software product in the form of a domain model; i.e. product specification. The information model describes concepts such as product offering, product specification, pricing information and domain rules. In particular, the information model used for the definition of the product modelling language is the Common Capability Model (CCM) defined by the British Telecom (BT) Group. The CCM describes common capabilities of BT's Matrix architecture and its portfolio package is a Unified Modelling Language (UML) Class Diagram that defines product specification concepts (Georgalas et. al., 2009).

The product-specific design tool is developed by following the tasks defined by the model-driven

Figure 4. Defining the product modelling language in the model-driven environment

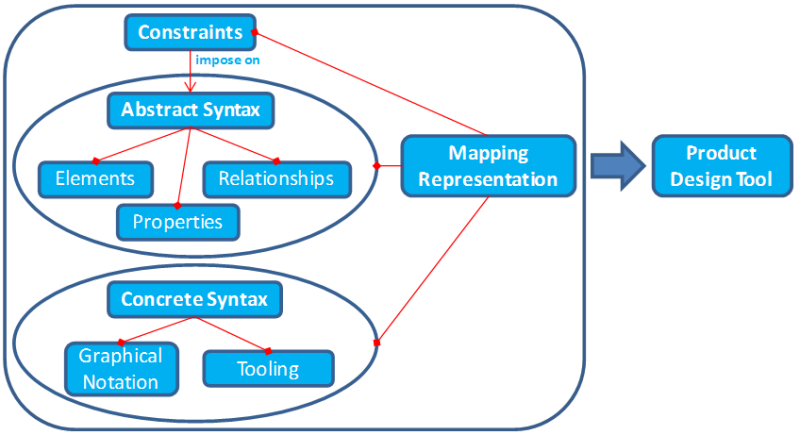


technique introduced in this chapter. Initially, product modelling concepts are derived directly from the existing UML Class Diagram of the CCM information model. Therefore, the primary task of requirement analysis is not executed since the concepts described in the product information model are taken for granted. The second task involves adapting the concepts of the CCM model to meet the expectations of the Ecore metamodelling language. This is straightforward since the artefacts defined within class diagrams are reasonably similar to metamodel artefacts. Therefore, the elements, relationships and properties of the product specification language are captured in the form of an Ecore metamodel. Figure 4 presents the product metamodel defined using the proposed model-driven environment, which defines the *Abstract Syntax* of the modelling language. The following task involves determining the rules that govern the product specification language and imposing, as illustrated in Figure 5, the required constraints onto the abstract syntax of the modelling language. This provides the capability to limit the designer input so as to avoid the definition of erroneous product models.

The definition of the abstract syntax and constraints is followed by the automated extraction of the concrete syntax of the modelling language. A suitable wizard allows the designer to select the product metamodel as the input model and fine-tune the model-to-model transformation by choosing the desired graphical notation for each metamodel artefact. The result obtained is an output model, called a *graphical metamodel*, which represents graphical objects such as rectangles, ellipses and connectors. In particular, the GMF component of the model-driven environment includes a visual library of objects from which the designer is able to select the desired ones in order to fine-tune the output graphical metamodel. Consequently, the graphical metamodel defines a mapping of the concepts of the modelling language to visual objects that allow representing the language concepts in a diagram.

Figure 5 illustrates that apart from the graphical notation, the *Concrete Syntax* of the modelling language includes also the necessary software tooling; *i.e. tooling metamodel*. The software tooling is obtained via an analogous wizard that allows mapping each metamodel artefact to the corresponding palette tooling of the product design tool to be generated. This step allows organising the concepts of the product specification language in separate groups of software tooling (i.e. buttons) on a palette. The palette is made available in the generated design tool and enables the drag-and-drop functionality, which allows designing the product model in the drawing canvas. Figure 4 illustrates

Figure 5. Developing the product design tool using the model-driven technique

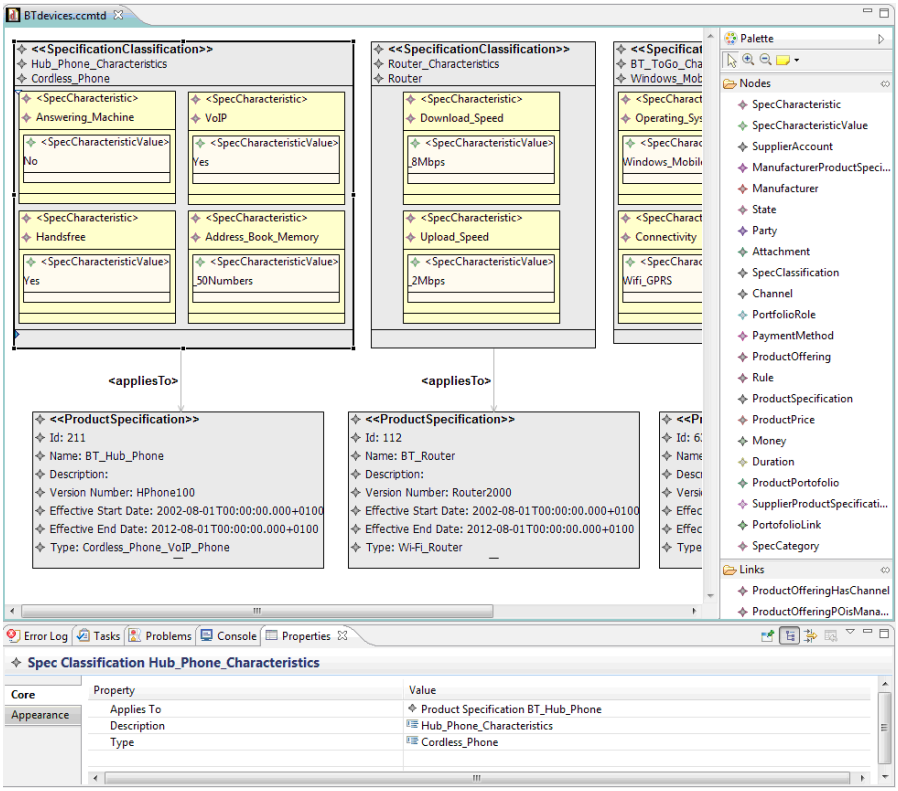


an analogous palette on the right hand-side of the figure, which refers to the software tooling of the Ecore metamodelling language. The generated product design tool resembles an equivalent modelling editor to the one illustrated in Figure 4, with the only difference that it incorporates the concepts of the product modelling language.

The subsequent task is optional since it allows customising the graphical and tooling metamodels in order to improve the presentation characteristics of the design tool. This is possible using the tree-based GMF editors that provide the capability to add, for instance, stereotypes (i.e. *labels*) to the visual objects that represent the language's concepts. Also the capability is provided to load icons for an artefact of the language instead of using graphical figures included within the GMF pool of visual objects. Further customisation capabilities are also provided in accordance to the requirements of the designer.

Having customised the concrete syntax of the language the software capability is provided that allows associating the artefacts of the product, graphical and tooling metamodels into a common *mapping representation*; i.e. *mapping metamodel*. For instance, an association describes how a metamodel concept (e.g. *"Specification" inFigure 4*) is mapped to the corresponding visual object (e.g. *rectangle figure*) and the respective palette tooling (i.e. *design tool palette button*). Therefore, the mapping defines all the necessary artefacts so as to facilitate the generation of the product design tool. This final task is actually an automated one since existing code generators are used to translate the mapping metamodel into an EMF-based Java implementation. As aforementioned the implementation of the design

Figure 6. Definition of the "BTEverywhere" software product using the product design tool

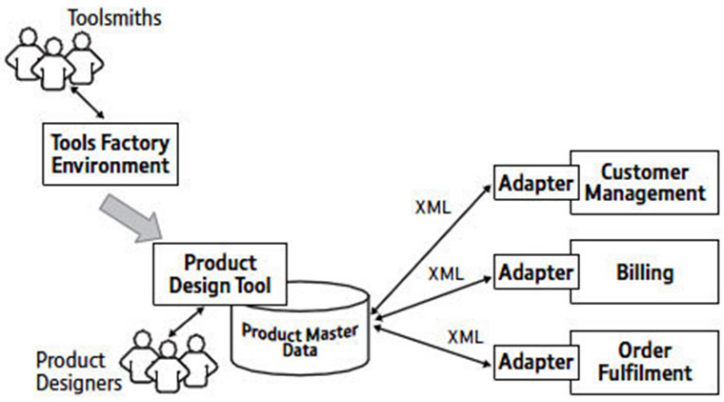


tool is delivered as Eclipse plug-ins, which are immediately integrateable and deployable as new capabilities of the environment.

The result is a product modelling tool (as illustrated in Figure 6) strictly dedicated to the abstract syntax, constraints, concrete syntax and semantics of the product modelling language. Figure 6 showcases an example domain model designed using the modelling tool that represents a software product called *"BTEverywhere"*, which provides the user with telephony, VoIP and broadband services. The software product actually offers the capability to shift seamlessly from the conventional telephony service while away from home to the VoIP service offered via broadband when located at home. This concludes the domain-specific modelling language development phase that delivers a fully-fledged product modelling tool to satisfy the designers and developers requirements. Hence, the proposed technique provides solutions to the aforementioned issues of the PLM process. It provides an enterprise-wide understanding of the software product, minimising the time and effort spend for interaction and product iterations amongst stakeholders and automates the process from concept inception all the way to product deployment.

Although the focus is on the development of design tools, we touch briefly how the product model is actually transformed into a fully-fledged deployable software product, so as to exemplify the end-to-end PLM-based development process. More details, on the transformation and the actual mapping can be found in (Georgalas et. al., 2009). Figure 7 illustrates at the top of the chain the *Toolsmiths* that are responsible to utilise the proposed *Model-Driven Environment* to generate the necessary *Product Design Tool*. *Product Designers* engage then with the definition of product models, which are subsequently transformed to *Product Master Data*. These data are captured in a specific format defined by an accepted enterprise-wide data model of the Master Data Management Platform (MDMP). Therefore, using the capabilities of the ATL and oAW frameworks of the model-driven environment the necessary data transformation scripts are defined that facilitate the transformation of product models to Product Master Data that populate respectively the MDMP repository. The generated product data captured in an XML format drive the configuration of OSS and BSS, so as to support the deployment of the new software product. Consequently, existing XML-based access interfaces defined in the form of adapters allow communicating Product Master Data to the OSS and BSS by transforming them to the system's native format as it flows to and from the MDMP. MDMP is the foundation for SOA capabilities across BT's Matrix architecture that makes OSS and BSS platforms data-driven

Figure 7. Master data management and PLM tooling driving OSS/BSS platforms (Georgalas et. al., 2009)



(Georgalas et. al., 2009). This removes laborious hard-coding tasks and maximizes data reuse in the PLM process.

Quantitative Evaluation using a Software Cost Estimation Model

This section examines the model-driven technique and environment introduced in this chapter and assesses their impact on software productivity. In particular, the capability to automate the development of domain-specific design tools is evaluated so as to determine the impact on software productivity. The evaluation examines the effect of the model-driven technique on the time, effort and cost required to develop the prototype product design tool using the following approaches:

1. Developing the product design tool using the proposed model-driven technique and its supporting intergrated Model Driven Environment (iMDE).
2. Developing the product design tool using the proposed model-driven technique and Borland Together 2008.
3. Manually implementing the product design tool from scratch without following any explicitly stated development process.

The evaluation is performed using the Post-Architecture model of COCOMO.II that allows estimating the *Effort in Person-Months (PM)* and the *Time to Develop (TDEV)* a software application taking into consideration an extensive set of parameters. Moreover, it provides the capability to estimate the necessary budget for developing the software application. The model considers the following inputs and defines the later equations, which allow deriving the effort and time to develop the software application.

1. The application's software size measured in thousand of lines of code (KLOC).
2. Five Scale Factors (SFs) that affect the development of the software application.

Seventeen Effort Multipliers (EMs) from which the TOOLS multiplier is divided and calibrated into three complementary (sub-) multipliers.

$$PM = A \times (KLOC)^{E} \times \prod_{i=1}^{17} \mathrm{EM}_i, \textit{ where } E = B + (0.01 \times \sum_{j=1}^{5} SF_i),$$

$$A = 2.94 \textit{ and } B = 0.91 \ (\textit{COCOMOII}.2000) \quad (1)$$

$$TDEV = C \times (PM)^{F}, \textit{ where } F = D + 0.2 \times (E - B),$$

$$C = 3.67 \textit{ and } D = 0.28 \ (\textit{COCOMOII}.2000) \quad (2)$$

Due to the importance of software tools in automating the development of design tools an extension of the model is considered (Baik et. al., 2002). The extension calibrates and divides the TOOLS multiplier into three complementary (sub-) multipliers, which are namely the completeness of Tool COVerage (TCOV), the degree of Tool INTegration (TINT) and the Tool MATurity (TMAT). These (sub-) multipliers are very important in the case of the model-driven technique since they describe important features of the model-driven environment that affect software productivity. In particular, the TCOV multiplier provides the capability to define and evaluate the coverage of activities undertaken in the software development process by the supporting tools. Furthermore, the TINT multiplier allows defining and evaluating the degree of integration of the tools used throughout the process and the effectiveness in achieving this integration. Finally, the TMAT multiplier allows stating and evaluating the maturity of the adopted toolset on the basis of the time it is used in the market and the technical support provided. This

Table 1. Satisfying the requirements for automating design tools development

	R1 - Abstract Syntax Definition	***R2 – Imposing Constraints***	***R3 - Concrete Syntax Definition***	***R4 – Design Tools Generation***
iMDE	MOF (EMF)	OCL	GMF	EMF, GMF
Borland Together 2008	MOF (EMF)	OCL	GMF	EMF, GMF

extension provides a more comprehensive estimate of the TOOL effort multiplier by calibrating the above (sub-) multipliers using the following equation (Baik et. al., 2002).

$$TOOL = 0.51 \times TCOV + 0.27 \times TINT + 0.22 \times TMAT \quad (3)$$

Prior to performing the calculations using the formal model, we assess the technique in a subjective manner against the requirements introduced in Section 1. Table 1 presents the software capabilities of the iMDE and Borland Together 2008, which satisfy the four necessary requirements for automating the development of design tools. The interesting point is that the same set of software capabilities is supplied by both environments for generating modelling tools. Firstly, the EMF provides a metamodelling language that conforms to the MOF standard and provides the capability to unambiguously define the abstract syntax of the modelling language. Secondly, the OCL specification is used as a common capability to impose the necessary rules that restrict the design of domain models. In addition, GMF facilitates the definition of the concrete syntax of the language and in conjunction with the EMF support the generation of design tools. It is important to point out that we have developed the prototype design tool using both environments in order to identify the differences in the development process. The dissimilarities identified are limited and have to do mainly with the enhanced graphical user interfaces provided by Borland Together, which eases to some extent the model-driven development tasks. Both environments provide though widely-used software capabilities that conform to the standards and support precisely the necessary development tasks.

Complementing the above subjective evaluation, we have utilised the software cost estimation model to carry out a quantitative assessment of the impact of the model-driven technique on software productivity. Note that the assessment is based on the assumption that developing the product design tool by manual coding, involves writing the same lines of code as in the case of the code generated for the design tool using the

Table 2. Rating scales for completeness of tool coverage

	TCOV
Very Low (1.17)	Text-Based Editor, Basic 3GL Compiler, etc.
Low (1.09)	Graphical Interactive Editor, Simple Design Language, etc.
Nominal (1.00)	Local Syntax Checking Editor, Standard Template Support, Document Generator, Simple Design Tools, etc.
High (0.9)	Local Semantics Checking Editor, Automatic Document Generator, Extended Design Tools, etc.
Very High (0.78)	Global Semantics Checking Editor, Tailorable Automatic Document Generator, Requirement Specification Aids and Analyser with Tracking Capability, etc.
Extra High (N/A)	Groupware Systems, Distributed Asynchronous Requirement Negotiation and Trade-off Tools, etc.

iMDE. Table 2 presents the ratings scales (Baik et. al., 2002) used to derive the TCOV multiplier, which serves as an example of how the rest of the ratings used in the calculations are derived. First, when the design tool is developed using the iMDE the TCOV rating is derived as *"HIGH"* (i.e. TCOV = 0.9) since the core components of the environment support most of the properties defined in this rating scale. For instance, automatic document generation is provided by the EMF component, extended design tools are also provided using the EMF and GMF components and local syntax checking by the GMF component.

By applying the same reasoning the TINT and TMAT sub-multiplier ratings are derived from the corresponding rating scales defined in (Baik et. al., 2002) and applied to Eq.3 to derive the TOOLS effort multiplier. For the iMDE the TINT rating is estimated as *"VERY HIGH"* (i.e. TINT = 0.78) due to the high degree of software tools integration, which is essentially provided by the plug-in architecture of the Eclipse platform. Finally, the TMAT rating is defined as *"VERY HIGH"* (i.e. TMAT = 0.78) due to the maturity of the environment's software tools (i.e. available in the market for more than three years) and the strong, large and experienced modelling community developing and/or using these modelling tools. Consequently, applying these individual sub-ratings in Eq. 3 the calibrated TOOLS rating for the case of using the iMDE is calculated as follows.

$TOOL_{iMDE} = 0.51 \times 0.9 + 0.27 \times 0.78 + 0.22 \times 0.78 \Rightarrow TOOL_{iMDE} = 0.8412$

$TOOL_{Borland} = 0.51 \times 0.78 + 0.27 \times 0.78 + 0.22 \times 0.78 \Rightarrow TOOL_{Borland} = 0.78$

$TOOL_{Coding} = 0.51 \times 1.17 + 0.27 \times 1 + 0.22 \times 0.78 \Rightarrow TOOL_{Coding} = 1.0383$

Using an analogous approach the individual sub-ratings and the calibrated TOOLS rating are calculated (as shown above) for the cases of using Borland Together 2008 and manual coding. For the case of Borland the individual sub-ratings are estimated as *TCOV=0.78, TINT=0.78* and *TMAT=0.78*. The only disparity has to do with the TCOV rating, which is estimated as *"VERY HIGH"*, mainly because of the enhanced front-end of the software tools provided by Borland that simplify the MDD tasks. Finally, in the case of manual coding the individual sub-ratings are estimated as *TCOV=1.17, TINT=1* and *TMAT=0.78*. The TCOV rating is estimated as *"VERY LOW"*, because text-based coding editors are used with basic 3GLs compilers, libraries and debuggers for creating manually the modelling tool; see Table 2. Furthermore, the integration of these software tools is relatively *"HIGH"* in development environments such as Netbeans and Eclipse and the maturity and competence of these software tools is *"VERY HIGH"*, since they are widely-used in the market for many years. Also a strong development and support group exists that evolves the capabilities of these software tools on a constant basis.

Apart from the TOOLS ratings, the ratings for the Scale Factors and the remaining Effort Multipliers included in COCOMO II are derived on the basis of the rating scales provided in (Boehm et. al., 2000). In this chapter, due to space limitations, we only discuss how one example multiplier is derived; i.e. SITE effort multiplier. This multiplier refers to multisite development (as defined by Boehm et. al., 2000) and determines if the members of the development team are collocated and if their communication is highly interactive or not. In the case of BT's development team the multiplier is rated as *"EXTRA HIGH"* (*i.e. SITE=0.80*). This is because the members of the team are collocated and their communication is highly interactive, since email, voice, video conferencing and other communication capabilities are provided. By applying analogous reasoning all individual ratings of the COCOMO II model are derived and applied to equations 1 and 2 to calculate the nominal effort and the time for developing the product design

tool. Therefore, using all the estimated ratings the calculations illustrated next are performed for the three individual cases described in this chapter.

(1) – MDD with Borland 2008

$E = 0.91 + [0.01 \times (3.72 + 2.03 + 4.24 + 1.1 + 1.56)] \Rightarrow E = 1.0365$

$PM_{Borland} = 2.94 \times (97.049)^{1.0365} \times [1 \times 0.9 \times (1 \times 1 \times 0.87 \times 1.17 \times 1.34) \times 1 \times 1.07\ 1 \times 1 \times 0.87 \times 0.85 \times 0.88 \times 0.9 \times 1 \times 0.91 \times 0.91 \times 0.78 \times 0.8 \times 1] \Rightarrow PM_{Borland} = 2.94 \times 114.69 \times 0.4 \Rightarrow$

$PM_{Borland} = 134.88$ Person-Months

$F = 0.28 + 0.2 \times (1.0365 - 0.91) \Rightarrow F = 0.28 + 0.2 \times 0.1265 \Rightarrow F = 0.3053$

$TDEV_{Borland} = 3.67 \times (134.8)^{0.3053} \Rightarrow TDEV_{Borland} = 16.4$ Months

(2) – MDD with the iMDE

$E = 0.91 + [0.01 \times (3.72 + 2.03 + 4.24 + 1.1 + 1.56)] \Rightarrow E = 1.0365$

$PM_{iMDE} = 2.94 \times (97.548)^{1.0365} \times [1 \times 0.9 \times (1 \times 1 \times 0.87 \times 1.17 \times 1.34) \times 1 \times 1.07\ 1 \times 1 \times 0.87 \times 0.85 \times 0.88 \times 0.9 \times 1 \times 0.91 \times 0.91 \times 0.8412 \times 0.8 \times 1] \Rightarrow PM_{iMDE} = 2.94 \times 115.29 \times 0.427 \Rightarrow$

$PM_{iMDE} = 144.73$ Person-Months

$F = 0.28 + 0.2 \times (1.0365 - 0.91) \Rightarrow F = 0.28 + 0.2 \times 0.1265 \Rightarrow F = 0.3053$

$TDEV_{iMDE} = 3.67 \times (144.73)^{0.3053} \Rightarrow TDEV_{iMDE} = 16.77$ Months

(3) – Manual Coding with IDEs

$E = 0.91 + [0.01 \times (3.72 + 2.03 + 4.24 + 1.1 + 1.56)] \Rightarrow E = 1.0365$

$PM_{Coding} = 2.94 \times (97.548)^{1.0365} \times [1 \times 0.9 \times (1 \times 1 \times 0.87 \times 1.17 \times 1.34) \times 1 \times 1.07\ 1 \times 1 \times 0.87 \times 0.85 \times 0.88 \times 0.9 \times 1 \times 0.91 \times 0.91 \times 1.0383 \times 0.8 \times 1] \Rightarrow PM_{Coding} = 2.94 \times 115.29 \times 0.53 \Rightarrow$

$PM_{Coding} = 179.65$ Person-Months

$F = 0.28 + 0.2 \times (1.0365 - 0.91) \Rightarrow F = 0.28 + 0.2 \times 0.1265 \Rightarrow F = 0.3053$

$TDEV_{Coding} = 3.67 \times (179.65)^{0.3053} \Rightarrow TDEV_{Coding} = 17.9$ Months

The above calculations illustrate that both the effort and time for developing the prototype design tool are decreased when highly competent model-driven environments are used. In contrast, implementing manually the product design tool increases noticeably the development effort and time. Consequently, this increase in effort and time results in a corresponding increase of the development costs. For instance, if we assume that the Average Monthly Work Rate (AMWR) is $1k then the development cost can be calculated for the individual cases using the following equation:

$$Cost = PM * AMWR \quad (4)$$

Therefore, the development of the product design tool using the iMDE and Borland Together 2008 incurs costs of $144.73k and $134.88k. On the contrary, higher costs are involved (i.e. $179.65k) when the design tool is implemented manually from scratch. The results depict clearly that the use of a competent model-driven environment that conveys to a systematic model-driven technique benefits the creation of design tools by reducing the development effort, time and cost.

Although the Post-Architecture model is widely-used and calibrated through data obtained from miscellaneous software projects, it still involves a degree of uncertainty and risk mainly due to its parametric inputs. In order to cope with these issues the evaluation introduces a complementary

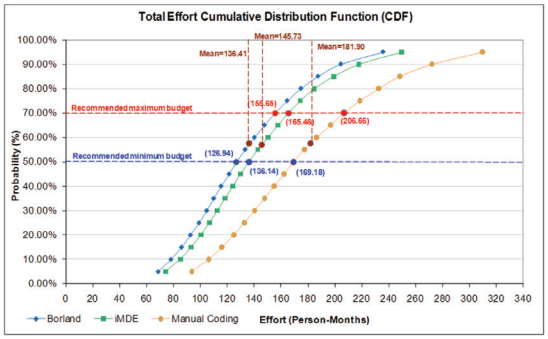
computational method that is based on the model. This is known as the Monte Carlo Simulation method that provides the capability to cope with the uncertainty and lack of knowledge involved when modelling phenomena such as the calculation of the effort, time and cost for the development of software design tools. The simulation is described as a method that computes samples within an input range and generates output data. These data define the probabilities that indicate if a software tool can be developed within a specific time frame and with a corresponding effort and budget involved. In particular, the application of the Monte Carlo Simulation method involves initially the definition of an estimated input range for each Scale Factor and Effort Multiplier using the Microsoft Excel Software Cost Analysis Tool (Lum & Monson 2003). These input ranges are also derived objectively on the basis of the rating scales presented in (Boehm et. al., 2000, Baik et. al., 2002). Hence, with the defined parameter ranges and the software size of the design tool as inputs the Analysis Tool executes a deterministic computation (i.e. using a mathematical formula). This generates a set of output data, which are aggregated into Cumulative Distribution Functions (CDFs) that represent respectively the effort and cost to develop the prototype design tool.

Figure 8 illustrates the CDF graphs generated by the Monte Carlo Simulation method that represent the corresponding effort and cost for developing the prototype design tool using the distinct development environments. The effort CDFs indicate clearly that for the set of computed probabilities the effort devoted to the development of the prototype is less when model-driven environments (i.e. iMDE, Borland Together 2008) are used. Furthermore, the costs CDFs illustrate that the development costs are correspondingly increased when the prototype design tool was manually developed from scratch using code-driven IDEs. In particular, for both CDFs the probable mean values computed are higher when manually developing the product design tool from scratch.

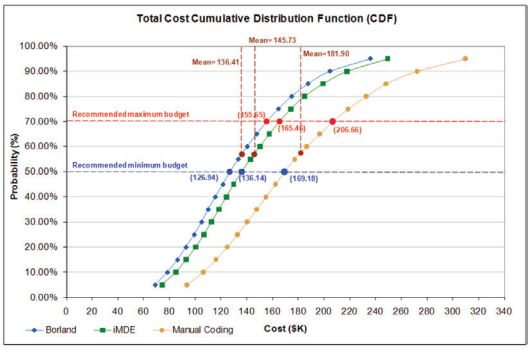
CONCLUSION

In this chapter we propose a model-driven technique and a supporting environment that demonstrate the benefits of employing MDD for automatically generating competent domain-specific design tools. The actual benefits are determined by a software cost estimation model that allows deriving the positive impact of the model-driven approach on software productivity. A prototype design tool is developed that forms the basis for assessing the impact of the approach with regards to the development effort, time and cost. Apart from the proposed model-driven environment (i.e.

Figure 8. Prototype design tool effort and cost cumulative distribution graphs



(a)



(b)

iMDE), an analogous environment (i.e. Borland Together 2008) is used to develop the prototype design tool. This reveals the necessity of using competent model-driven environments that adhere to a systematic model-driven technique for automating the development of domain-specific design tools. Furthermore, the environments conformance to the MDA specifications is established as another fundamental requirement that enables better understanding of the defined modelling languages and facilitates rapid adoption of the developed domain-specific design tools.

The development of the product design tool using the iMDE and/or Borland Together showcased a reduction in development overheads; i.e. effort, time and cost. In particular, the use of these environments provided an increased automation in software generation, reducing the overheads to a greater extent than what the software cost analysis results indicate; i.e. expected effort is reduced by 19.88%. Nevertheless, the estimated analysis results are suggestive of the positive impact of advanced model-driven tools in rapidly and unambiguously developing domain-specific design tools. For instance, the Figure 8 illustrates that the mean nominal effort is equal to *PM=181.90 Person-Months* and the mean cost is equal to *Cost=$181.90k* when manually implementing the product design tool. In contrast, the nominal effort and cost are significantly reduced when developing the design tool using the iMDE or Borland Together as illustrated also clearly in Figure 8.

As part of future work the extension and/or calibration of the software cost estimation model, so as to address parameters (i.e. Effort Multipliers) that are closely correlated to model-driven software development, will enable the optimisation of the quantitative evaluation method introduced in this chapter. For example, the TOOLS effort multiplier can be extended to include *Code Generation* as a sub-multiplier that affects significantly the estimation on software productivity for model-driven techniques.

REFERENCES

Achilleos, A., Georgalas, N., & Yang, K. (2007). *An open source domain-specific tools framework to support model driven development of OSS*. In ECMDA-FA, (LNCS 4530), (pp. 1 – 16).

Achilleos, A., Yang, K., & Georgalas, N. (2008). A model-driven approach to generate service creation environments. In *Proceedings of the IEEE Globecom, Global Telecommunications Conference,* (pp. 1–6).

Achilleos, A., Yang, K., Georgalas, N., & Azmoodeh, M. (2008). Pervasive dervice vreation using a model driven Petri Net based approach. In *Proceedings of the IEEE International Wireless Communications and Mobile Computing Conference (IWCMC)*, (pp. 309-314).

Afonso, M., Vogel, R., & Texeira, J. (2006). From code centric to model centric software engineering: Practical case study of MDD infusion in a systems integration company. In *Proceedings of the Workshop on Model-Based Development of Computer-Based Systems and International Workshop on Model-Based Methodologies for Pervasive and Embedded Software*, (pp.125-134).

Baik, J., Boehm, B., & Stecee, B. M. (2002). Disaggregating and calibrating the CASE tool variable in COCOMO 2. *IEEE Transactions on Software Engineering*, *28*(11), 1009–1022. doi:10.1109/TSE.2002.1049401

Balasubramanian, K., Gokhale, A., Karsai, G., Sztipanovits, J., & Neema, S. (2006). *Developing applications using model-driven design environments. IEEE Computer*. Vanderbilt University.

Boehm, B., Abts, C., Clark, B., Devnani-Chulani, S., Horowitz, E., & Madachy, R. (2000). *COCOMO 2 model definition manual, version 2.1*. Center for Systems and Software Engineering, University of Southern California.

Borland Together Integrated and Agile Design Solutions. (2006). Getting started guide for Borland Together 2006 for Eclipse. Retrieved from http://techpubs.borland.com/together/tec2006/en/GettingStarted.pdf

Chen, Z., Boehm, B., Menzies, T., & Port, D. (2005). Finding the right data for software cost modelling. *IEEE Software*, *22*(6), 38–46. doi:10.1109/MS.2005.151

Chonacky, N. (2009). A modern Tower of Babel. *Computing in Science & Engineering*, *11*(3), 80. doi:10.1109/MCSE.2009.45

Clark, T., Evans, A., Sammut, P., & Willans, J. (2004). An eXecutable metamodelling facility for domain-specific language design. In *Proceedings of the Object-Oriented Programming, Systems, Languages, and Applications Workshop on Domain-Specific Modelling*.

Deursen, A. V., Klint, P., & Visser, J. (2000). Domain-specific anguages: An annotated bibliography. *ACM SIGPLAN Notices*, *35*(6), 26–36.

Emerson, J. M., & Sztipanovits, J. (2004). Implementing a MOF-based metamodelling environment using graph transformations. In *Proceedings of the 4th OOPSLA Workshop on Domain-Specific Modeling*. Retrieved from http://www.dsmforum.org/events/DSM04/emerson.pdf

Evermann, J., & Wand, Y. (2005). Toward formalizing domain modelling semantics in language syntax. *IEEE Transactions on Software Engineering*, *31*(1), 21–37. doi:10.1109/TSE.2005.15

Frankel, D. S. (2003). *Model driven architecture: Applying MDA to enterprise computing*. Indianapolis: Wiley Publishing Inc.

Freudenthal, M. (2009). Domain-specific languages in a customs Information System. *IEEE Software*, *99*(1), 1–17.

Georgalas, N., Achilleos, A., Freskos, V., & Economou, D. (2009). Agile product lifecycle management for service delivery frameworks: History, architecture and tools. *BT Technology Journal*, *26*(2).

Georgalas, N., Ou, S., Azmoodeh, M., & Yang, K. (2007). Towards a model-driven approach for ontology-based context-aware application development: A case study. In *Proceedings of the IEEE 4th International Workshop on Model-based Methodologies for Pervasive and Embedded Software (MOMPES)*, (pp. 21-32).

Gerber, A., & Raymond, K. (2003). MOF to EMF: There and back again. In *Proceedings of the OOPSLA Workshop on Eclipse Technology eXchange,* (pp. 60 – 64).

Graaf, B., & Deursen, A. V. (2007). Visualisation of domain-specific modelling languages using UML. In *Proceedings of the Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, (pp. 586-595).

IBM. (2009). *Eclipse platform technical overview*. Retrieved from http://www.eclipse.org/whitepapers/eclipse-overview.pdf

Iscoe, N., Williams, G. B., & Arango, G. (1991). Domain modelling for software engineering. In *Proceedings of the IEEE International Conference on Software Engineering*, (pp. 340-343).

Kelly, S., & Pohjonen, R. (2009). Worst practices for domain-specific modelling. *IEEE Software*, *26*(4), 22–29. doi:10.1109/MS.2009.109

Kleppe, A., Warmer, J., & Bast, W. (2005). *MDA explained: The model driven architecture, practice and promise*. Boston: Addison-Wesley.

Ledeczi, A., Bakay, A., Maroti, M., Volgyesi, P., Nordstrom, G., & Springle, J. (2001). Composing domain-specific design environments. *IEEE Computer*, *34*(11), 44–51.

Lum, K., & Monson, E. (2003). *Software cost analysis tool user document*. California: NASA-Jet Propulsion Laboratory Pasadena.

Mohamed, M., Romdhani, M., & Ghedira, K. (2007). EMF-MOF alignment. In *Proceedings of the 3rd International Conference on Autonomic and Autonomous Systems,* (pp. 1 – 6).

Molnár, Z., Balasubramanian, D., & Lédeczi, A. (2007). *An introduction to the generic modelling environment.* Model-driven development tool implementers forum. Retrieved from http://www.dsmforum.org/events/MDD-TIF07/GME.2.pdf

Nytun, J. P., Prinz, A., & Tveit, M. S. (2006). Automatic generation of modelling tools. *In Proceedings of the European Conference on Model-Driven Architecture, Foundations and Applications (ECMDA-FA)* (LNCS 4066), (pp. 268-283).

OMG. (2003). *Model Driven Architecture (MDA) specification guide v1.0.1.* Retrieved from http://www.omg.org/docs/omg/03-06-01.pdf

OMG. (2005). *Meta Object Facility (MOF) core specification v2.0*. Retrieved from http://www.omg.org/docs/formal/06-01-01.pdf.

OMG. (2005). Object Constraint Language (OCL) specification v2.0. Retrieved from http://www.omg.org/docs/formal/06-05-01.pdf

Santos, L. A., Koskimies, K., & Lopes, A. (2008). Automated domain-specific modeling languages for generating framework-based applications. In *Proceedings of the 12th International Conference on Software Product Lines*, (pp. 149-158).

Sprinkle, J., Mernik, M., Tolvanen, J.-P., & Spinellis, D. (2009). What kinds of nails need a domain-specific hammer? *IEEE Software*, *26*(4), 15–18. doi:10.1109/MS.2009.92

Staron, M. (2006). Adopting model driven software development in industry-a case study at two companies. In *Proceedings of the International Conference on Model Driven Engineering Languages and Systems,* (LNCS 4199), (pp. 57-72).

Wirth, N. (2008). A brief history of software engineering. *IEEE Annals of the History of Computing*, *30*(3), 32–39. doi:10.1109/MAHC.2008.33

Zbib, R., Jain, A., Bassu, D., & Agrawal, H. (2006). Generating domain-specific graphical modelling editors from metamodels. In *Proceedings of the Annual IEEE Computer Software and Applications Conference*, (pp. 129-138).

KEY TERMS AND DEFINITIONS

Domain Specific Language(s): A modelling/specification or programming language(s) that describes a specific problem domain and can be used to design domain specific models.

Domain Specific Modelling: Describes a process that raises the level of abstraction by introducing domain models as the prime entities in software development.

Metamodelling: The process that guides the definition of a metamodel, which describes the elements, properties and relationships of a particular modelling domain; i.e. domain specific language.

Model-Driven Development: A software development methodology that focuses on the design and implementation of software applications at an abstract platform-independent level.

Software Cost Model: A mathematical model that provides the capability to estimate/calculate the required time, effort and cost to develop software applications.

Software Productivity: Defines the measure of efficiency, which can be described in terms of time, effort and cost required for the development of software applications.

Software Service Creation: Describes a software development process that deals with the analysis, design, validation and implementation of software services.