Universitat Politècnica de Catalunya
Departament de Llenguatges i Sistemes Informàtics
Màster en Computació

Tesi de Màster

Considering
Non-Functional Requirements
in
Model-Driven Engineering

Estudiant: David Ameller
Director: Xavier Franch

Data: 22/06/2009

Meta-Document section

Table of contents

Table of figures

Table of tables

Master thesis scope

Our research group, *Grup de recerca en Enginyeria del Software per als Sistemes d'Informació* (GESSI) [1], that belongs to the *Llenguatges i Sistemes Informàtics* (LSI) [2] department of the *Universitat Politècnica de Catalunya* (UPC) [3], is working on several research lines. One of the newest adopted research line is Model-Driven Engineering (MDE) [4], others such as Requirements Engineering (RE) [5] and Software Quality (SQ) [6] are already consolidated research lines in our group.

I am principally working in MDE, but with the influence of RE and SQ. My research is focused on the design of a framework called Responsibility Detection and Transformation (RDT). This framework will take into account Non-Functional Requirements (NFRs) to take decisions over the architecture and the technologies of the resultant software product. For more information of my previous work in this direction see [7-11].

Master thesis structure

The Master Thesis is structured in three parts:

- The first part is a systematic review of MDE, and concretely the research over NFRs in MDE. It is divided into six sections: first, the review method used; second, an introduction to MDE; third, the main research topics on MDE; fourth, the research done for NFRs in MDE; and finally, sections 5 and 6 are the analysis and the conclusions respectively.
- The second part is an overview of the architectural aspects of Responsibility Detection and Transformation (RDT). It is divided into two sections: first, an introduction to RDT, and a comparison between RDT and Model-Driven Development (MDD); and second, the description of RDT architecture.
- The third part is the construction of a survey for IT companies and organizations with the aim to analyze its knowledge about MDE and the importance that the companies and organizations grant to NFRs.

Preface

Model-Driven Engineering (MDE) is the result of an emergent research area that uses models as the principal artifact in the engineering processes. This initiative is based on the separation of the essential specification of the system and its implementation using a specific platform.

MDE represents a new leap in software engineering and development, especially for information systems. MDE changes radically the way we are used to develop software because the principal artifact of work are models, there are still programming languages, but the work tends to be done in a conceptual and visual way.

By using the MDE ideas we can beneficiate of a higher abstraction level and improved platform independence. A clear example of these benefits is the adaptation to new technologies; this problem can be solved or alleviated by using technology-independent models that can be transformed semi-automatically into technology-specific models that fulfill the trendy technologic needs.

From the point of view of analyzing and evaluating all the possible architectural and technological alternatives, Non-Functional Requirements (NFRs) play a crucial role to take correct decisions over the architectural styles and selection of the implementation technologies. Unfortunately, as explained in this Master Thesis, the importance of NFRs is not reflected in the current MDE research.

Motivation

Requirements engineering is necessary for software development, the most famous software development methods have a specific step to elicit or treat with requirements, e.g., Rational Unified Process (RUP), iterative development, spiral model, waterfall model, etc.

According to Glinz [12], requirements set the boundaries of an important dimension of the software, its quality. For example, in the ISO 9126 [13] quality model, we can find quality characteristics like: functionality, reliability, usability, efficiency, maintainability, and portability. These quality characteristics are related with the habitual NFR specifications. E.g. “the system should be available 99% of the time”. This NFR is related with the reliability of the system.

The reason of the selection of NFRs in MDE topic for this Master Thesis, as many recognized authors said [14], is that it is not feasible to produce a software product that satisfies the stakeholder needs without taking into account NFRs.

My hypothesis is that “having good support for NFRs inside MDE is one of the missing key-factors that would lead to the success of MDE”.

Part I

Systematic Review:

Non-Functional Requirements

in

Model-Driven Engineering

1. Review method

The reason to perform this systematic review is to provide a background and identify gaps in current MDE research in order find areas for further investigation.

This systematic review is based on B. Kitchenham's methodology [15]. She proposes systematic review guidelines specific for software engineering researchers. Her proposal defines a systematic review as follows:

"A systematic review is a means of evaluating and interpreting all available research relevant to a particular research question, topic area, or phenomenon of interest. Systematic reviews aim to present a fair evaluation of a research topic by using a trustworthy, rigorous, and auditable methodology."

The proposal is based on similar guidelines for medical researchers with the aim of reusing the knowledge and experience of a well-known and consolidated research area. Her main intention with the guidelines is to introduce the concept of rigorous reviews of current empirical evidence to the software engineering community.

There are several features that differentiate a systematic review from a conventional literature review:

- *Definition of the review protocol*. Review protocol specifies the research question being addressed and the methods that will be used to perform the review.
- *Definition of the search strategy*. Search strategy aims to detect as much of the relevant literature as possible.
- *Documented searches*, so that readers can assess its rigor and completeness.
- *Explicit inclusion and exclusion criteria* to assess each potential primary study (a systematic review is a secondary study based on primary studies).
- Systematic reviews specify the information to be obtained from each primary study including quality criteria by which to evaluate each primary study.
- A systematic review is a prerequisite for quantitative meta-analysis.

1.1. Review protocol

As stated in [15], *"A review protocol specifies the methods that will be used to undertake a specific systematic review. A pre-defined protocol is necessary to reduce the possibility researcher bias."*

The following points are the ones indicated in the guidelines for the development of a review protocol:

1. The research question. It is intended by this systematic review to answer a question that identifies and/or scopes future research activities. The research question of this systematic review is:

   Are Non-Functional Requirements correctly supported by the current state of Model-Driven Engineering?

It is important to define what is “correctly supported” in order to procure a good answer to this question. I understand that “correctly supported” means that:

- All kinds of NFRs should be specifiable, preferably in a commonly accepted notation.
- The produced software with a MDE process should be compliant with NFRs specifications.

The answer to this question will be interesting to software engineers and, in concrete, to MDE experienced engineers.

To answer the research question, first, I must get a full idea of the current MDE state of the art, to accomplish with this objective I will explore all its relevant research topics (section 3). Second, I will focus on the works done for NFRs inside MDE (section 4), this objective will be accomplished with a deeper search (deeper search is described in section 1.3.1). Then with all this knowledge I will propose a framework to support NFRs in MDE, documented in the second part of this Master Thesis.

2. The strategy used to search for primary studies and the selection criteria is described in section 1.3.
3. Study quality assessment checklists and procedures. Due to the diverse nature (methods, processes, languages, tools, etc.) of the consulted works, define a quality assessment procedure for all of them would not result on any significant information, so for this Master Thesis, this part of the systematic review method is omitted.
4. Data extraction strategy. Each research topic studied has a specific data extraction. The data extraction strategy followed for all of them consists in a selection of the most representative common attributes, and a comparison between the studied works. Each research topic ends with a summarizing table with the specific data extraction.
5. Synthesis of the extracted data. In section 5 of this Master Thesis the analysis of the examined documents is detailed.
6. Project timetable. There is no timetable for this work apart from the deadlines imposed by the Master Thesis. This work will be refined till the end of the Master Thesis.

1.2. Information sources

Two kinds of information sources are being considered: specific to MDE, and software engineering in general (see Figure 1). By specific, I mean that only MDE related topics are treated in the information source, and, by generic, I mean that MDE and other topics can be treated. In the generic sources I only searched for the MDE related works.

As first selection criteria, workshops, book, and electronic sources are only considered when they are specialized in MDE.

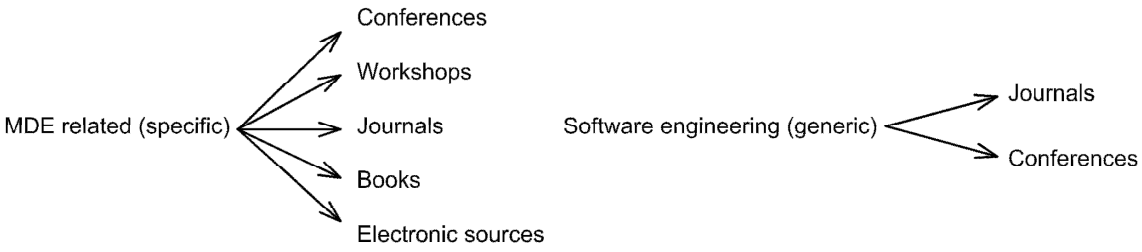

Figure 1: Information sources

1.2.1. Conferences

A great number of the works revised belong to conferences. Four kinds of conferences were considered: MDE specific, MDE related, generic, and others. MDE specific conferences were examined in depth, while related ones, generic ones, and others were only consulted in specific tracks. The selection of the conferences was done by consulting experts and by comparing the statistical information shown in Table 1.

Conferences specific to MDE:

- *ECMDA*: European Conference on Model-Driven Architecture
- *ICMT*: International Conference on Model Transformation
- *MoDELS*: International Conference on Model-Driven Engineering Languages and Systems (previously UML)

Conferences related to MDE:

- *ECOOP*: European Conference on Object-Oriented Programming
- *ER:* International Conference on Conceptual Modeling
- *OOPSLA*: Object-Oriented Programming, Systems, Languages & Applications

Conferences of software engineering:

- *ESEC/FSE*: European Software Engineering Conference/Foundations of Software Engineering
- *ICSE*: International Conference on Software Engineering

Other conferences:

- *ECSA*: European Conference on Software Architecture
- *RE:* IEEE International Requirements Engineering Conference
- *TOOLS*: Tools International Conference

In Table 1 is shown a comparison between all these conferences, taking into account the rank (based on the Australian Ranking of ICT Conferences, 2007), the number of editions, the publisher, and the acceptance of the recent editions (* means that the conference was not celebrated in that

year). Note that some of these conferences are relatively new, for example ICMT is in its first edition (previously it was a workshop), and ECMDA is in its fourth. This fact can be understood as: "MDE is getting more relevance in the research community".

	Rank	# Editions	Publisher	% Acceptance			
				2005	2006	2007	2008
ECMDA	?	4	Springer	28/82 (34%)	30/78 (39%)	16/60 (27%)	31/85 (37%)
ICMT	?	1	Springer	*	*	*	17/48 (35%)
MoDELS	A	11	Springer	46/170 (27%)	51/178 (29%)	45/158 (28%)	58/271 (21%)
ECOOP	A	22	Springer	24/172 (14%)	21/160 (13%)	25/160 (16%)	27/138 (20%)
ER	B	27	Springer	31/169 (18%)	37/158 (23%)	37/167 (22%)	33/178 (19%)
OOPSLA	A+	23	ACM	32/174 (18%)	26/157 (17%)	33/156 (21%)	33/117 (28%)
ESEC/FSE	B/A	16	ACM	32/201 (16%)	25/125 (20%)	43/251 (17%)	31/152 (20%)
ICSE	A+	30	ACM/IEEE	44/313 (14%)	36/395 (9%)	49/334 (15%)	56/371 (15%)
ECSA	?	2	Springer	*	*	18/62 (29%)	23/83 (28%)
RE	A	16	IEEE	35/175 (20%)	42/181 (23%)	29/172 (17%)	38/164 (23%)
TOOLS	B	46	Springer	*	*	24/78 (31%)	21/58 (36%)

Table 1: List of conferences

The selection of the more relevant MDE research topics was inferred during the search done in the MDE related conferences by looking at the topics of the calls for papers.

1.2.2. Workshops

There are many workshops related to MDE, many appear and disappear in few years. The interesting thing is that the presented works normally are more innovative or "blending-edge", as contrast these works are many times immature or incomplete. The selection of the workshops was done by consulting experts and electronic sources (see section 1.2.5).

This is the list of consulted workshops:

- DSML: Domain-specific Modeling Languages
- GraMoT: Graph and Model Transformation
- DSDM: Desarrollo de Software Dirigido por Modelos (Spanish)
- DSADR: Domain Specific Analysis and Design for Reuse
- MoDSE: Model-Driven Software Evolution
- MoDISE-EUS: Model-Driven Information Systems Engineering: Enterprise, User and System Models
- MoDeVVA: Model-Driven Engineering, Verification, and Validation: Integrating Verification and Validation in MDE (previously MoDeVa)

A special mention has to be done to DSDM, a Spanish workshop in which I have attended in the last two editions (DSDM'07 and DSDM'08). This workshop was very interesting for me because there I met almost all the Spanish research groups related to MDE and model-driven initiatives in general.

1.2.3. Journals

Another important reference for documentation has been journals in which the published works are supposed to be widely accepted by the community.

The selection of the journals was done by consulting experts and by comparing the statistical information shown in Table 2.

	Rank	Total Cites	Impact Factor	Immediacy Index	Articles 2007
ACM Trans. Softw. Eng. Methodol.	**A***	534	2.792	0.417	12
Computer IEEE	**B**	1874	1.367	0.236	106
IEEE Software	**B**	1446	1.462	0.161	62
IEEE Trans. Softw. Eng.	**A***	3672	2.105	0.385	52
Journal of Object Technology (JOT)	**B**	no statistical information available			
Software and Systems Modeling	**B**	no statistical information available			
Software Practice and Experience	**A**	809	0.542	0.082	61

Table 2: List of journals

In Table 2 is shown a comparison between the selected journals, the rank has been obtained from the Australian Ranking of journals (July 2008) and the statistical information has been obtained from the Journal Citation Reports (JCR Science Edition 2007). The Impact Factor is the relation between *Cites to recent articles* and *Number of recent articles*, the Immediacy Index is the relation between *Cites to articles from 2007* and *Number of articles in 2007*.

One Spanish journal was consulted, in concrete nº 192 (March-April 2008) of Novatica, which includes a monographic about Model-Driven Development (MDD).

1.2.4. Books

Books are important information sources to get the basic notions of the concepts behind MDE, in Table 3 is a list of the books I had read during the elaboration of this systematic review (the number of citations are taken from Google™ Scholar).

The selection of the books was done by personal recommendations and by the number of citations. A huge part of the introduction to MDE was done with the information provided by these books.

Ref.	Book	Citations (date)
[16]	***MDA Distilled: Principles of Model-Driven Architecture***	**246 (May, 2009)**
[17]	***Model-Driven Architecture in Practice***	**30 (May, 2009)**
[18]	***Executable UML***	**496 (May, 2009)**
[19]	***Model-Driven Architecture: Applying MDA to Enterprise Computing***	**567 (May, 2009)**
[20]	***MDA Explained: Practice and Promise***	**809 (May, 2009)**
[21]	***Model-Driven Software Development: Integrating Quality Assurance***	**3 (May, 2009)**

Table 3: List of books

1.2.5. Electronic sources

Some of the information mentioned in this document does not belong to any published source. Some of these electronic sources have been obtained doing searches over the web for related topics, other are recommendations of the author, or community groups. In Table 4 are shown the relevant electronic sources consulted:

Ref.	Book	Description
[22]	***DSDM, MDA y Aplicaciones (Spanish)***	**Community Web site**
[23]	***MOdeling LAnguages***	**Web portal**
[24]	***Model-Driven Inside***	**Web portal**
[25]	***Model Transformation***	**Community Web site**
[26]	***DSM Forum***	**Community Web site**
[27]	***Metamodel.com***	**Web portal**
[28]	***The history of conceptual modeling***	**University project**

Table 4: List of electronic sources

1.3. Search and data extraction strategy

The search and data extraction strategy used for conferences, workshops, and journals has six steps:

1. *Selection of tracks*: Only the MDE related tracks were examined (only for conferences).
2. *Selection of articles based on their titles*: To try to reduce the number of articles, I ruled out the ones that evidently are out of the scope of this systematic review.
3. *Read the abstract and find keywords*: Many times the titles are confusing or not representative enough. Reading the abstract helps to refine the selection done in step 1. Also, keywords were useful for the classification.
4. *Classification inside one research topic*: Each work was classified inside one research topic, or as generic MDE information.
5. *Read the article*: Only the selected works by the steps 1 and 2 were read in depth. The reading of the article was made marking the relevant parts and annotating comments to ensure that future readings will take less time.
6. *Mark its relevance*: as final step, I have annotated a personal view of the article, pointing out the benefits, and the detected disadvantages. Also, in my opinion, I have marked the articles in three grades depending on its relevance.

1.3.1. Deeper search

For my research topic, NFRs in MDE, I have made a deeper study. First, keyword searches for the most referenced papers in IEEE Xplore, ACM Portal, SpringerLink, and Google Scholar. The searches were firstly done in September 2008 and repeated every two months.

The keywords used were:

("model-driven architecture" OR
"model-driven development" OR
"model-driven engineering") AND
("non-functional requirements" OR
"non-functional properties")

Second, having identified the relevant papers, I have performed a backward and forward reference search, for the author and for the cited papers.

Some of the papers of the deeper search are from more than two years ago, but I have selected them for completeness and/or because its contributions are still valid.

2. Introduction to Model-Driven Engineering

This section introduces the Model-Driven Engineering (MDE) and related initiatives. It is divided into five parts: history of software development, terminology, methodology, points of view, and research lines.

2.1. History of software development

With this historical review I am trying to show that it is reasonable to think that MDE is a good candidate to be the next established way to develop software, and that MDE would take more relevance in the near future. To do so, I have explored the previous changes in the tendencies of software development, and looked for similarities in the current state of MDE. Similar thought is shared by one of the most influential articles:

"Model-driven development holds promise of being the first true generational leap in software development since the introduction of the compiler." – B. Selic, IEEE Software, 2003. [29].

Software development has been a very changing topic taking into account that it is a relatively young discipline. Each new leap of software development is characterized by an improvement in the abstraction level that tends to approach the development methodologies to the human-thinking, and, in consequence, the platform independence, that makes the software more portable. This evolution is shown in Figure 2.

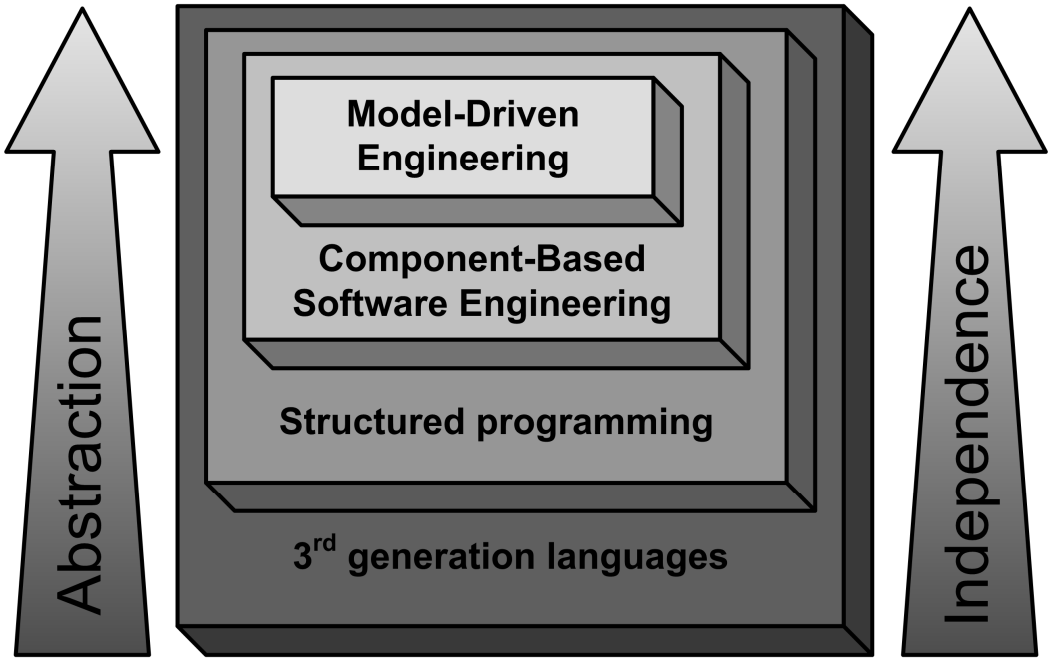

Figure 2: Evolution of software development

2.1.1. First leap: 3rd generation languages

The first general-purpose electronic computers (e.g., ENIAC, 1946) were programmed using 1st and 2nd generation languages (machine code and assembly languages), but the beginning of the software development for information systems is situated in the 1960s with the common usage of the 3rd generation languages (3GL), such as: FORTRAN (1957), LISP (1958),

COBOL (1959), or BASIC (1964). These languages represent a significant advance on the platform independence and the abstraction level.

It is interesting to see that 3GLs usage instead of assembly languages had some similarities with what happens today with Model-Driven Development (MDD). Assembly programmers did not believe that compilers could produce assembly code of the same quality as humans can. This may be true, but only for very short programs, and the same happens with MDD in a greater scale.

2.1.2. Second leap: Structured programming

In the later 1960s a new tendency begins with the famous E. W. Dijkstra letter published in the Communications of the ACM (1968), he argued that GOTO statements should be eliminated from all higher level programming languages. Some years later, in 1972, E. W. Dijkstra et al. publishes the book "*Structured programming*". The intensive usage during the 1980s of structured programming methodology and the appearance of languages such as PASCAL (1970), C (1972), or ADA (1983) establishes this period of time as a leap on software development.

2.1.3. Third leap: Component-Based Software Engineering

Component-Based Software Engineering (CBSE) started in the latter 1970s as one of the responses to the "software crisis":

"[The major cause of the software crisis is] that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem." – E. Dijkstra, "*The Humble Programmer*", ACM Turing Award Lectures, 1972.

In the 1980s, modular programming added concepts like interface or library. The majority of the structured languages mentioned before had support for modular programming or had been extended to support it. Modula-2 (1978) is a clear example of a language originated by this new tendency.

Object-Oriented (OO) paradigm can be seen as an evolution of the first ideas proposed in CBSE (B. J. Cox, "*Object oriented programming: an evolutionary approach*", 1986). OO began in the earlier 1960s as an educational paradigm, but the first OO programming language, Smalltalk, was not released until 1980 (the standardized version, ANSI Smalltalk, was not released until 1998). This paradigm has gained the attention of the software development community in the later 1990s till nowadays. Today, OO paradigm is considered the common way to develop software for information systems. With its success some new languages have appeared (e.g., Java, 1995), but it is more important to notice that all the relevant languages from the previous leaps have been adapted in some way to support object-orientation (e.g., C++, 1983; ADA95, 1995).

Another component-based approach that, nowadays, is getting more relevance is CBSE applied to Distributed Computing, CORBA (1991) and DCOM (1996) were two of the first contributions that tried to facilitate the use of objects in Distributed Computing software, currently SOAP (1998) and REST (2000) have substituted them. Today, the common way to develop software for distributed computing is using Web Services. Web Services

were proposed by W3C organization, and extended by OASIS with standards and architectures such as Service-Oriented Architecture (SOA).

2.1.4. Fourth leap: Model-Driven Engineering

In the beginning the use of models for software development did not consider the ideas of automatic or semi-automatic development. One of the most transcendent model-based languages was Entity-Relation (ER) [30] proposed by P. P. Chen in 1976. The following statement resumes the importance of the ER contribution:

"Entities and relationships are a natural way to organize physical things as well as information ... The ER concept is the basic fundamental principle for conceptual modeling. It has been with us since thousands of years ago and will be with us for many years to come." P. P. Chen, SIGMOD Record, March 2004.

The conceptual modeling was an active research topic during the 1980s, some approaches were published to extend ER (e.g., EER, 1989). The bases of UML appeared in 1991-1992 with the contributions of G. Booch (Booch Method), J. Rumbaugh (OMT) and I. Jacobson (OOSE). Twenty-one years after ER, in 1997, the first version of UML was presented (UML 2.0 dates from 2005). Today, UML is used as the standard model-based language for software development. ER and UML are the roots of MDE. MDE was born in the earlier years of 2000s with the launch of Model-Driven Architecture (MDA, 2001) [31], and a progressive unification of initiatives that take models as a principal element in software development.

Languages of 4^{th} (e.g., SQL, 1974) and 5^{th} (e.g., Prolog, 1972) generation (4GL and 5GL) have a representation inside MDE. Domain Specific Languages (DSL) are the normal evolution of 4GL, and constraints-based languages like Object Constraint Language (OCL, 1999 [32]) have a clear correspondence with 5GL.

Nowadays, MDE continues getting more and more relevance each year, but it is still in incubation time due to two principal indicators: new languages are still appearing to cover detected needs in MDE (e.g., transformation languages described in sections 2.3 and 3.4), and there are few (or none) initiatives for adapting the previous leaps to support MDE (e.g., JMI [33] can be considered as a step in this direction).

Comparing the previous leaps with the current state of MDE we can think that MDE could be the next leap in software development.

2.1.5. Historical analysis

The timeline in Figure 3 shows the four software development leaps considered in this section. Analyzing them we can infer the following observations:

- All of them have an incubation time (grey part on the left) in which the research has the most significant part, and new languages and ideas tend to come out, it is noticeable that this incubation time tends to be longer in each new leap.
- All of them have a period of time of intensive usage (the darkest part). This time also tends to be longer in each leap.

- All of them have a transition time (grey part on the right) where the most relevant languages and ideas tend to be adapted to support the next leap ideas.

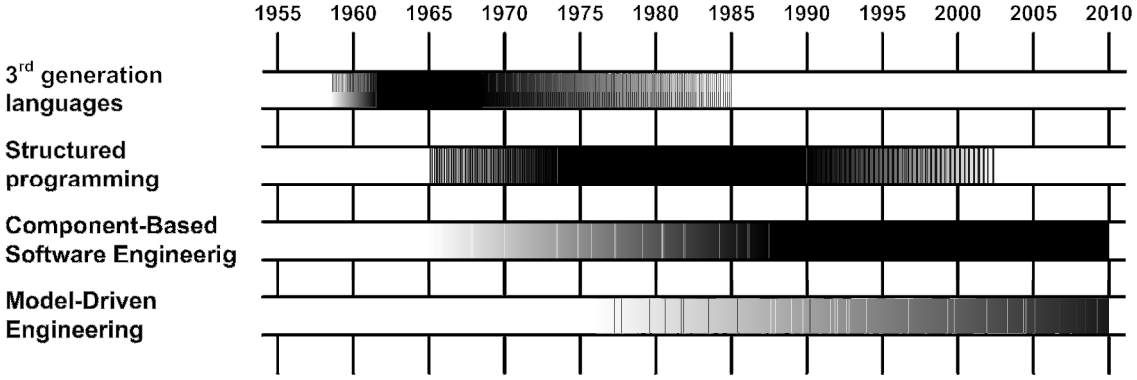

Figure 3: Timeline evolution of software development

The most important thing to notice in this brief historical view is that each leap absorbs the previous improvements, and adds some structural methodology of development that brings a higher abstraction level. In other words, it is a matter of fact that MDE uses some kind of CBSE (normally OO paradigm), which uses structured programming (at least for code generation), and therefore 3GL.

2.2. MDE terminology

All the model-driven initiatives have lots of new terminology with a particular meaning. Most of the definitions used in this section are extracted from the glossaries of [16, 20, 31], some of them have been slightly adapted.

2.2.1. Initiatives

Model-driven initiatives itself are named by acronyms, the following are the most representative:

- *Model-Driven Architecture (MDA)*: The first white paper from OMG referring to MDA was published in 2000. Later, in 2003 the current version of MDA guide [31] was published. All model-driven initiatives follow the principle that "everything is a model", stated in [34].

  In a plenary session celebrated in Montreal between the 23rd and 26th of August, 2004, the following definition of MDA [35] was accorded.

  "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."

- *Model-Driven Development (MDD)*: In 2003 an article with the same name as the initiative [36] is published in IEEE Software. The article defines MDD as:

  "*Model-driven development is simply the notion that we can construct a model of a system that we can then transform into the real thing.*"

  If we use this definition with a relaxed definition of model (see section 2.2.2), developing software with 3rd generation languages is already MDD, so all developers are MDD developers.

  One difference with MDA is that MDD is not adhered to any of the OMG standards, as M. Fowler said in [37], but the main contribution of MDD is the flexibility offered to define development processes.

- *Model-Driven Engineering (MDE)*: three years later, in 2006, another article also with the same name as the initiative [4] is published in the IEEE Computer Society. The article defines MDE as:

  "Model-driven engineering technologies offer a promising approach to address the inability of third-generation languages to alleviate the complexity of platforms and express domain concepts effectively."

  By this definition we can understand MDE as the evolution of CASE tools. The MDE initiative proposes a wider processes definitions (not limited to development as MDD), and support for model analysis to take decisions or simply for reasoning.

In Figure 4 the evolution of these three initiatives is represented. Note that the MDD and MDE initiatives have appeared earlier than the mentioned year. The years shown in the figure are refereed to the first publication in a journal.

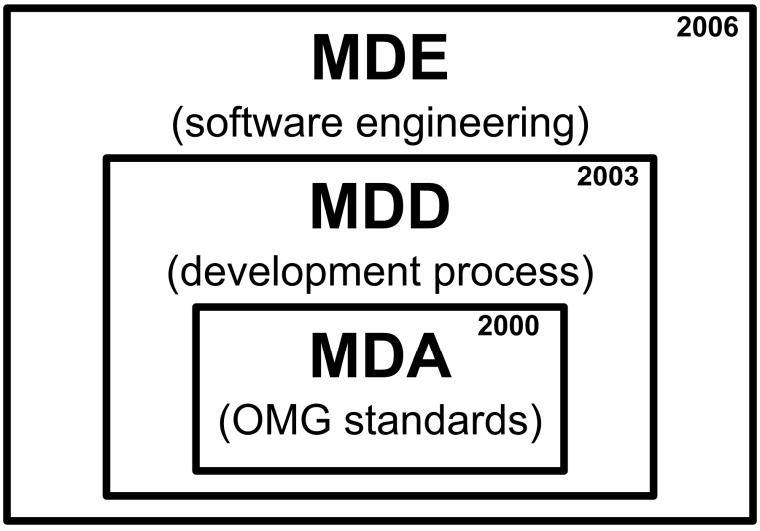

Figure 4: Representation of the model-driven initiatives

There are other similar initiatives that are based on models. For example, Conceptual Schema-Centric Development (CSCD) [38] is an initiative that centers its efforts on the conceptual schema.

2.2.2. Models

The term *model* is one of the most used, but its meaning can sensibly change between contexts (widely explained in [34, 39, 40]). In Software Engineering the term model traditionally refers to an artifact formulated in

some modeling language, this definition is too narrow for MDE where the code of a program can be interpreted as a model in some cases. In the book "*MDA Explained*" [20] the term model is defined as: "*A description of (part of) a system written in a well-defined language*". With the last definitions we can accept code as a model.

In a more generic viewpoint, we can take the definition of the term model from Merriam-Webster's Online Dictionary (March, 2009) "*an exact representation of something in greatly reduced size*" or the definition of the Cambridge Dictionary "*a representation of something, either as a physical object which is usually smaller than the real object, or as a simple description of the object which might be used in calculations*" this definition adds an important point to think about, a model is a simplification. This is not always true in MDE because in some approaches it is pretended to have executable models. For me an executable model can not be a simplification because it needs all the knowledge of the software system that is representing in order to be executable.

In the first MDA white paper the three most important kinds of models for MDE were defined, these three kinds of models refers to the development stages of software going from the problem space to the implementation solution:

- *Computation Independent Model (CIM)*: A CIM is a view of the system from the computation independent viewpoint. CIM is many times called business model.

  An example of CIM could be the model that represents the process to deliver a package (simplified):

  1. An employee takes the package from the client's home and takes it to the nearer office.
  2. The package is transported to the nearest office of the addressee.
  3. An employee takes the package to the addressee.
- *Platform Independent Model (PIM)*: A model that contains no details that have meaning only within a specific platform. This kind of model does not have relation with any implementation technology. Following the previous example, the PIM model will only contain the parts that are supposed to be made by the computer (simplified):
  1. The system assigns an identifier to the packet.
  2. The system calculates the most economic route to deliver the packet.
- *Platform Specific Model (PSM)*: A model that contains details that have meaning only within a specific platform. This kind of model normally has a relation with some implementation technologies. In difference with the previous example, in this case the PSM model will contain the same parts of the process, but specified with a concrete technology (simplified):
  1. The system uses a trigger of the Oracle database to generate an identifier for each package.
  2. The system calculates the most economic route to deliver the packet using the web services of the contracted airlines.

2.2.3. Four-layers metamodeling architecture

In Figure 5 (left) the four-layers metamodeling architecture is shown (explained in [41, 42]), and (right), as an example, the representation for UML models.

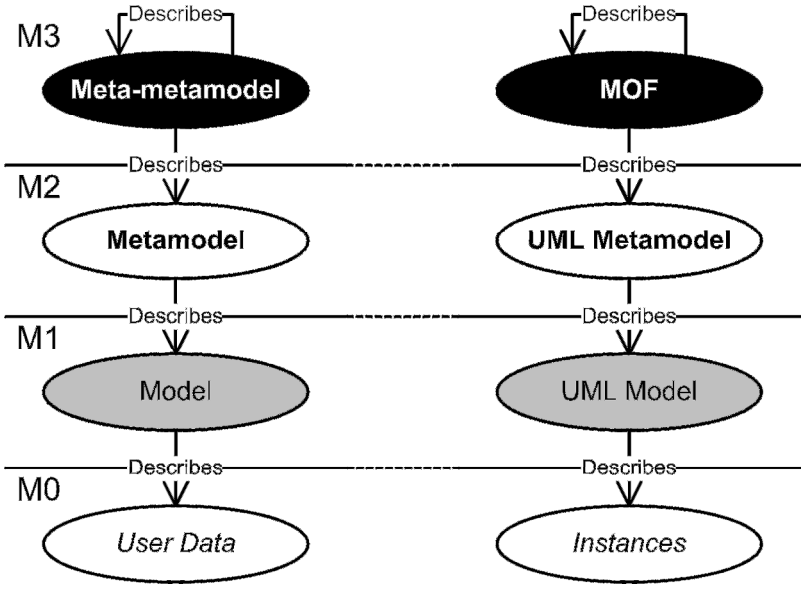

Figure 5: Four-layer metamodel architecture

- *M3*: The layer that contains meta-meta-metadata that describes the properties that meta-metadata can exhibit. If there were a M4 layer it would look just the same as M3, because M3 is self-describing. Meta Object Facility (MOF) is a standard from OMG [43]. MOF is a language to describe metamodels. To have an approximate idea of what is MOF, we could think in UML limited to class diagrams. Some implementations of MOF are JMI [33] and Ecore [44].
- *M2*: The layer that contains the meta-metadata that describes the properties that metadata may exhibit (for example, UML elements such as Class, Attribute, and Operation). Here we find metamodels, a description or definition of a well-defined language in the form of a model.
- *M1*: The layer that contains the metadata of the application (for example, the classes of an object-oriented system, or the table definitions of a relational database).
- *M0*: The layer that contains the data of the application (for example, the instances populating an object-oriented system at runtime, or rows in relational database tables).

In [42], C. Atkinson and T. Kühne describe a more advanced structure based on the separation of linguistics and ontology. Some of these ideas are currently used in MOF 2.0.

2.3. Methodology

The usual methodology for Model-Driven Development (MDD) is a transformation process as stated in many publications (p.8 [20], p.2 [45], etc.). It is shown in the slashed box of Figure 6:

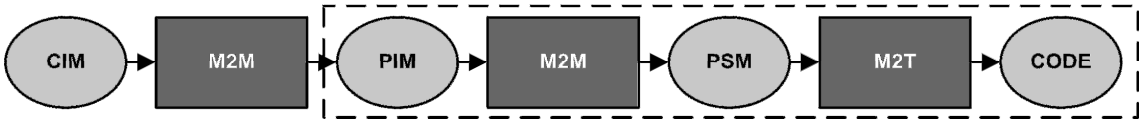

Figure 6: Common transformation flow

The boxes are transformations. The boxes labeled with M2M are referring to the acronym Model to Model transformation and the one labeled with M2T is referring to Model to Text transformation. Transformations are a key aspect in MDE because we can reuse the work done in a transformation for all models. For instance we could build a transformation to obtain a normalized UML model from a non-normalized UML model, and reuse this transformation for every non-normalized UML model.

The transformation between CIM and PIM is not very common in the publications I have read, it is shown here for completeness. An example of this kind of transformations is in [46], in this work the CIM is a goal-oriented model, the PIM is a multidimensional data warehouse model and the PSM is a specific data base technology model.

The transformations between models (M2M) are defined with transformation languages. Currently the most used transformation languages are: QVT [43], ATL [47], RubyTL [48, 49], MTF [50], and VIATRA2 [51] (for a further look to transformation languages approaches see [52] and for more information on this research area see section 3.4). In Figure 7 the transformation between models within the four-layer metamodel architecture is shown. The transformation is defined in the M2 level by specifying a mapping between the elements involved in the transformation and the transformation is applied to the M1 level models.

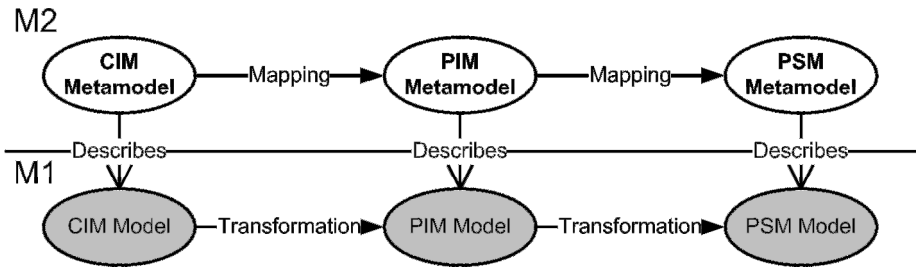

Figure 7: Transformation between models in layers M1 and M2

The transformations from a model to a text (M2T) file are normally done by some scripting or template-based languages, these are some examples: Velocity [53], MTL [54], Xpand [55, 56], JET [54].

2.4. Points of view

In MDD there are basically two points of view (see [57] for details) that affect the transformation flow with many implications on the way the software is developed:

- *Translationist*: the development flow is only from models to code as shown in Figure 6 and Figure 8. This approach is defended by S. J. Mellor, in particular with his book named “Executable UML” [18]. Note that normally the code generated is not complete, so it will be modified by hand at the end of the MDD process.

- *Elaborationist*: the development flow is done in two directions, so we are able to regenerate the models when some change is done in the code as shown in Figure 8. This approach is defended by A. Kleppe, in particular with his book named "MDA Explained" [20].

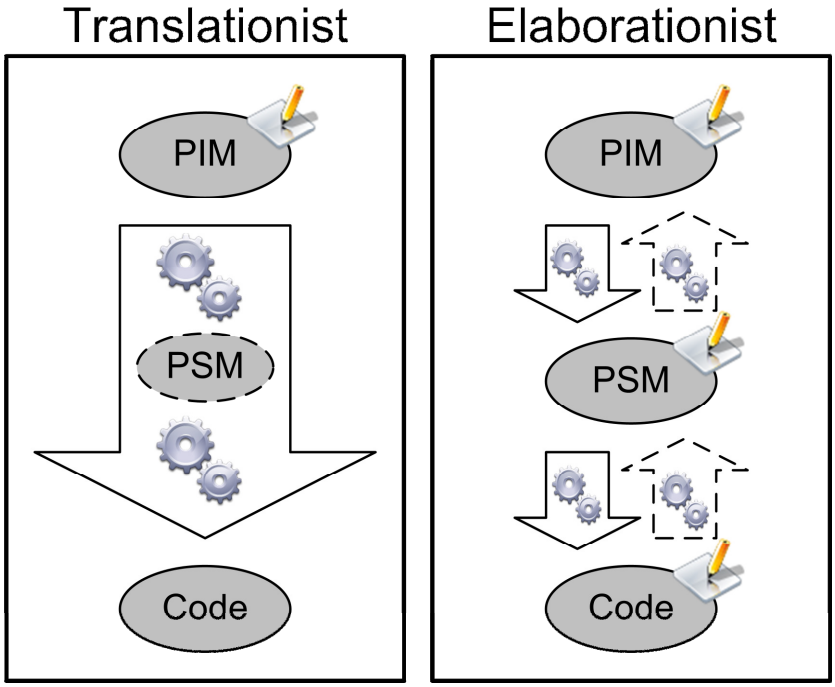

Figure 8: Translationist and Elaborationist approaches[1]

With the boom of the model-driven initiatives, especially with MDD, there have appeared a set of very introspective questions opening interesting points of discussion about MDD and its capabilities, some of them expressed in [58] (see the "ugly" section) others are taken from books and conference talks.

> *Should a MDD process generate the 100% of the source code? Or should the software development be based completely on models?*

In general, there is not a final response, but there are some works for a domain specific that almost accomplish with these objectives (see section 3.2). It is necessary a very specific domain, and a very restricted group of applications to be able to generate the full source code.

> *Should a MDD process be fully automatic?*

In [45] is said, based on the experience, that the MDD is not an automatic process, it needs some interaction (this fact is also mentioned in section 7.2 of [20]).

> *Are the Model-Driven initiatives too good to be true?*

About this question the opinions are very confronted, some allegations in favor of model-driven initiatives are said in section 12 of [20].

[1] The pencil means that the model can be modified during the process

3. Main MDE research topics

Model-Driven Engineering (MDE) is not a new idea, but we are currently experiencing a fast evolution of the technologies that support it, that is why many emergent research topics are centered on MDE research area. The research topics below have been selected to have a full overview of MDE state of the art. This selection of topics was done by looking at the calls for papers topics of the MDE related conferences.

- Frameworks and CASE tools: This topic is not really a research topic, but researchers are very dependent on its features (normally there is no time to develop a full new system).
- *Domain-Specific Modeling*: This topic has medium and small projects that normally solve one or few domain specific problems. Many times these projects are ad-hoc.
- *Methodologies*: Apart from the generic development methodology, MDD, explained in section 2.3, new methodologies of development and engineering have appeared to solve or alleviate problems.
- *Transformation languages*: This research topic is important because all we can do related to the model-driven initiatives is limited to the expressiveness of the transformation languages.
- *Correctness*: Because models tend to be part of the product instead of the documentation, we must bring facilities to ensure that the models are correct.

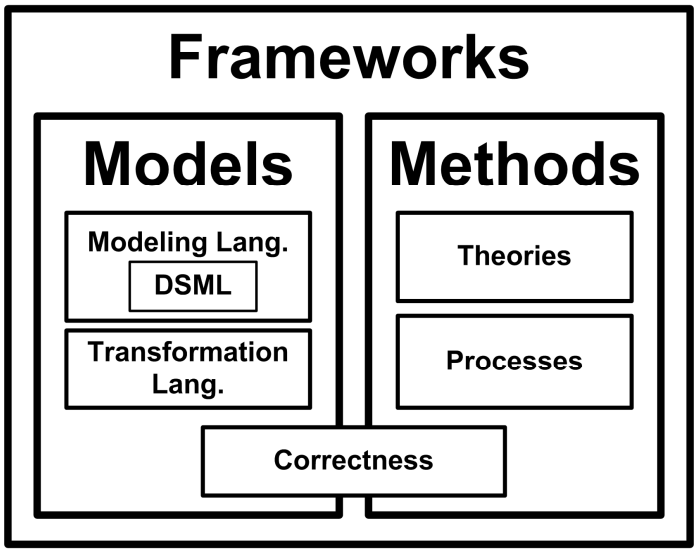

Figure 9: Main MDE research topics

Important contributions to these research topics are detailed in the next sections.

3.1. Frameworks and CASE tools

In this section generic frameworks and CASE tools are studied. Frameworks are normally big projects that support extension mechanisms, and they are able to drive all necessary development steps, while CASE tools are oriented to solve a particular step of development, or to offer specific facilities to the software developer.

3.1.1. Frameworks

The most popular development frameworks are:

- *Eclipse Modeling Project (EMP)* [59]: EMP is a unified set of model-based development technologies, modeling facilities, tooling, and standards implementations. Eclipse is an open source project that, at the time of writing, is in version named "Ganymede".
- *AndroMDA* [60]: AndroMDA is open source MDA generator. It is used to generate from simple CRUD applications to complex enterprise applications. AndroMDA comes with ready-to-use plug-ins for common architectures like Spring, EJB, .NET, Hibernate, and Struts.
- *Oslo* [61]: It is the code-name of an ongoing project driven by Microsoft. Oslo is a modeling platform that, I suppose, will be included in the next major revision of Microsoft Visual Studio. Currently, it is under development.

Other frameworks are Eclipse-based. They are customized versions of Eclipse that are more than an extension, they are a full remake. Remarkable ones are:

- *openArchitectureWare (oAW)* [55]: It is a modular MDA/MDD generator framework implemented in Java. It supports parsing of arbitrary models, and a language family to check and transform models as well as generate code based on them. Supporting editors are based on the Eclipse platform.
- OptimalJ: Its first release, in 2001, was for Sun Microsystems' NetBeans IDE. Since 2006, it was an Eclipse integrated environment. OptimalJ was used to develop J2EE applications based on MDA specifications using implementation and technology patterns. In 2008, Compuware decided to discontinue it.
- *Rational Software Architect* (RSA) [62]: RSA is the evolution of the Rational Rose line of products that was acquired by IBM in 2003. RSA is the IBM response to MDD; this framework is also related with Rational Unified Process (RUP).

The majority of the framework web sites states that they are currently used on enterprises.

The open source frameworks are thought to be extended with plug-ins. This fact is a key benefit for the model-driven initiatives because it facilitates the development of a solution for a concrete technology, a concrete architectural style, or a specific domain.

Table 5 contains a comparison of the current and active versions, at the time of writing, of the principal frameworks.

	Interface	M2M	M2T	UML	Metamodels	Model validation
EMP Ganymede	Eclipse	ATL/QVT	JET	1.x/2.x	EMF	Java
AndroMDA 3.3	None	Java	Velocity	1.4/2.0	MOF XMI	OCL/Java
oAW 4.3	Eclipse	Xtend	Xpand	1.x/2.0	EMF	Check
RSArchitect 7.5	Eclipse	Own-made	Own-made	2.x	Own-made	Unknow

Table 5: Comparison between frameworks

The flexibility offered by EMP being part of the Eclipse project is one of the reasons that makes it the first spot for the research community while AndroMDA that uses its own system is less attractive. AndroMDA is supposed to be compatible with Eclipse in its next major version, as oAW and others do. Unfortunately, AndroMDA project had very low activity in the last year, so it is not clear if they will finally publish a next major version soon.

3.1.2. CASE tools

Many MDE CASE tools are the result of research prototypes for some specific topic. As an example, USE [63] is a tool that has been developed to validate models with its constraints.

In this systematic review I focus on CASE tools that are model editors because they are often used on MDE projects. Eclipse as a whole offer facilities to construct model editors with technologies like EMF [64], this is the case of MOSKitt [65], but many model editors are standalone pieces of software. The most used editors are:

- *MagicDraw* [66]: It is a software and system modeling tool designed for software analysts and programmers. It is, as shown in Table 6, the best conceived model editor, but the problem is that it is commercial software and a bit expensive.
- *Poseidon for UML* [67]: Poseidon for UML is a CASE tool used to create models with the Unified Modeling Language (UML). It was originated from the ArgoUML project, but massive changes were necessary in order to make ArgoUML a commercial project; as result the two efforts are very different. Lately, the company behind Poseidon, Gentleware, has developed a new Eclipse tool for modeling called Apollo.
- *Rational Rose* [68]: It was developed by Rational Software and sold to IBM in 2003. Currently, it has been discontinued in favor to RSA.
- *MOSKitt* [65]: It is a free CASE tool developed by a governmental institution of Valencia (Spain) to support a methodology named gvMètrica.

Table 6 contains a comparison of the current and active versions, at the time of writing, of the principal model editors.

	IDE	MDD integration	Reverse engineering	XMI support	UML
MagicDraw 16.0	Standalone, Eclipse & others	Yes	Yes	Yes	2.0
Poseidon 6.0	Standalone	Little	Yes	Yes	2.0
Apollo 3.0	Standalone & Eclipse	Little	Yes	Yes	2.0
MOSKitt 0.5	Eclipse	Yes	No	Yes	2.0

Table 6: Comparison between model editors

3.2. Domain-Specific Modeling

Many tools are oriented to solve concrete problems (for example, obtain the database schema from a UML model) or to give support to a Domain-Specific Modeling Language (DSML). Domain-Specific Languages (DSL) are

languages with very specific goals in design and implementation. A domain-specific language can be either a visual diagramming language, such as the Eclipse Modeling Framework (EMF), or textual languages. This section is centered in the visual ones.

MDE is applicable to very different domains. In this situation try to embrace many domains would be too complex. On the other hand, when we focus our efforts on a specific domain using a DSL we can, principally, take advantage on two aspects:

- We can offer facilities to the end user because we are closer to the problem domain. An example of these facilities appears in WSNAD [69], where the use of a domain-specific metamodel and Platform Specific Model (PSM) representation brings clear understanding of the problem for the specialists on Wireless Sensor Networks.
- We can know the constraints of the domain. This allows us to make a concrete validation or verification of the models. An example of this appear in ClassSheets [70], where a new way of object-oriented modeling of spreadsheets is proposed to reduce the error-prone.

In many cases the solution proposed is a full integrated environment to perform specific tasks, an example could be REMM-Studio [71], this environment is specific for modeling software requirements that are also validated with its metamodel. Another interesting point is that REMM-Studio was built to generate documentation instead of code.

In Table 7 the characteristics of the three projects mentioned in this section are resumed. It is noticeable that domain specific solutions have several problems:

- They are ad-hoc solutions, so they are not reusable.
- They can not be easily integrated with a more complex process of development.

	Facilities offered	Validation	Reusable	Integration
WSNAD	Clear understanding for the domain users	Domain specific	No	No
ClassSheets	Reduce error-prone	Domain specific	No	No
REMM-Studio	Requirements engineering oriented	Domain specific	No	No

Table 7: Comparison between domain specific approaches

3.3. Methodologies

The complexity of MDE requires support tools and methodologies. There are two kinds of methodologies, methodologies to resolve a specific problem inside a full development methodology (e.g. in [72] a methodology to generate operations to create and remove elements from a conceptual schema is explained) and methodologies that contemplates all the development process (e.g. i^2MAP [73] is a methodology that reminds us that MDD is not limited to UML models, in this case with the aim to reduce errors in the software development the process uses two kinds of models: goal-oriented and UML).

In this section two methodologies one for general development (OO-Method) and one for the integration of information systems to MDA (SIM) are explained in more detail.

3.3.1. OO-Method

OO-Method [17] is a methodology that covers many of the weakest parts of the typical MDD methodology. OO-Method is divided in 3 parts: "Requirements Engineering", "Conceptual Modeling", and "Software Representation". The first two represents the problem space while the third is in the solution space. This methodology follows the principles stated in "*Executable UML"* [18], in other words it is about doing the development on the models. OO-Method uses four kinds of models:

- *Object Model*, this model defines relations between the classes of the problem domain.
- *Dynamic Model*, this model defines possible sequences of services for the objects of a given class and aspects related with the interaction among objects.
- *Functional Model*, this model captures the semantics associated to object state changes, triggered by events.
- *Presentation Model*, this model captures the information that defines the characteristics of the user interface, and the way in which users will interact with the system.

All these models are specified using a formal language called OASIS (Open and Active Specification of Information Systems). This language is based on standards like XML and DTD.

OO-Method includes the generation of the software product. It is done by applying an abstract execution model which includes a strategy for code generation that determines which software representation corresponds to each conceptual pattern.

3.3.2. Systems Integration Methodology (SIM)

Systems Integration Methodology [74] is proposed for the integration of information systems to be used with model-driven initiatives, this methodology consists in five steps:

1. *Study of the technology and architecture*: All the available information is obtained and studied. The information is checked, validated, and developed.
2. *Develop use cases*: Previously developed platform-specific applications are obtained and analyzed to define the functions required. The goal is to identify structures and components susceptible to be considered separately, along with abstractions and general concepts.
3. *Creation of metamodel*: UML Profiles are used to define metamodels. The basic principle for defining each profile is to obtain generalizations between different programming languages, platforms, and technologies, as well as to incorporate other relevant aspects related to the integration of inherited systems and applications.
4. *Plug-in construction*: Having defined the functions and the metamodel, we can begin to construct the MDD framework plug-in, whose function is to direct the compilation and packaging of the model exported in XMI.
5. *Metamodels Unification*: The creation of plug-ins and metamodels for each specific domain should provide enough information to define a shared and unified metamodel.

Appling this methodology we can convert a heterogeneous information system into a MDA system with all the advantages that this connotes (portability, adaptability, reusability, etc.).

3.3.3. Comparison of methodologies

In Table 8 the characteristics of the three methodologies mentioned in this section that contemplates the full development process are resumed.

	Domain	Principal objective	Principal benefit	Principal disadvantage	Steps	Flexibility	Real case studies
OO-Method	Information systems	Software development	The resulting models are executable	OASIS dependent	3	None is descrived	Yes
SIM	Information systems	Systems integration	The resulting system has the benefits of MDA	Specific technology dependent	5	Cartridge based	Yes
I²MAP	Embedded systems	Software development	Reduce unsatisfactory requirements	Concrete domain applications	3	None is descrived	unknown

Table 8: Comparison between methodologies

3.4. Transformation languages

Transformations are the cornerstone of model-driven initiatives. That is the reason why the versatility of a model-driven approach is limited to the expressiveness of the transformation language. In [75], J. Muñoz et al. presents a classification of the transformation languages depending on its capabilities.

The transformation processes can be more complex than the perspective shown in Figure 6, for example in [76] there is a M2M transformation from PIM to PIM, the objective of this transformation is to derive the software architecture from the analysis model following the RUP. Another example is the M2T transformation shown in [77], it generates code from the PIM to C# without instantiating the PSM.

The number of transformations grows with the complexity of the model-driven process, [78] and [79] proposes environments to describe the compositions of transformations that supports the use of different transformation tools and languages. Taxonomies for model transformation applications are described in [78] and in [80].

As said in the introduction, transformations between models (M2M) are defined with transformation languages. Common transformation languages are: QVT [43], ATL [47], RubyTL [48, 49], MTF [50], VIATRA2 [51], and Xtend [55]. M2T transformations are normally done with some scripting or template-based language, like: Velocity [53], MTL [54], Xpand [55, 56], and JET [54]. A comparison between all these languages is shown on Table 9, the symbol “?” means that the required information is not documented and the symbol “-” means that the characteristic is not applicable to the transformation language.

		Standards compliance	Declarative (what)	Imperative (how)	Endogenous (same metamodel)	Exogenous (diff.metamodels)	Multiple Sources	Multiple Targets	Horizontal (e.g.PIM2PIM)	Vertical (e.g.PIM2PSM)	Bidirectional transformations
M2M	ATL 2.0	✓	✓	✓	✓	✓	✓	✓	✓	✓	×
	MTF	×	✓	×	✓	?	×	×	✓	✓	✓
	QVTR	✓	✓	×	✓	?	?	?	?	?	×
	RubyTL	?	?	?	?	?	?	?	?	?	?
	VIATRA2	×	✓	✓	✓	?	?	?	?	?	?
	Xtend	×	×	✓	✓	?	?	?	✓	✓	×
M2T	JET	×	×	✓	–	–	✓	✓	–	–	?
	MTL	✓	?	?	–	–	?	?	–	–	?
	Velocity	×	×	✓	–	–	✓	✓	–	–	×
	Xpand	×	×	✓	–	–	✓	✓	–	–	?

Table 9: Comparison between transformation languages

As we can see in Table 9, bidirectional transformations are not supported in general, but they are getting more importance because many researchers are working in this direction and not only in MDE area, visualization and database researchers are also interested in bidirectional transformations. The QVT specification [43] is supposed to support bidirectional transformations, but its current implementations do not support it. The most promising solution to this problem is the use of Triple Graph Grammars (TGG) [81]. Currently there is a set of tools, MOFLON [82], that supports TGG.

3.5. Correctness

Correctness is an important topic in software engineering. The following are some of the related problems that can be found in the common operations done in the MDE processes:

- *Model Transformation*: Loss of information
- *Model Validation*: Unsatisfied constraints

Other basic operations applicable to models are: *merging* (join two or more models), *matching* (look for similarities between models), *diff* (look for differences between models), *slice* (partial view of a model) and *split* (separate a model in several models). For more information on these operations see [83]. These basic operations can have problems like: duplication of some elements or none/multiple matches for some elements.

All these problems can be solved or alleviated using validation and verification methods.

Testing is the common way of validation in software engineering and MDE makes no difference. Therefore, MDE emphasizes on validation techniques

based on rule languages. These rules are normally specified inside of the domain-specific metamodels. As mentioned before in section 3.2, one of the benefits of using DSL is that we can validate models with a more restricted validation rules, the domain specific rules plus the generic modeling rules.

These rules can be seen as constraints, in this case Object Constraint Language (OCL) [32, 84] is the principal reference. OCL has been accepted by the community as the standard way to specify the textual constraints in UML models.

As in other cases, OCL have supporters and detractors. Some studied lacks of OCL are:

- OCL does not provide primitives to specify invariability (what can and what must not be changed when a method is executed), a study to this problem is presented in [85]. This is important to validate a model with OCL constraints.
- OCL is highly error-prone and it is difficult to understand, possible solutions for this problem are:
  - Use predefined constraints as exposed in [86], this also improves the capability to do automatic actions with model-driven tools.
  - Derive the operations contracts automatically as exposed in [87].

The proposed solutions seem to be insufficient for MDE because some specific languages have appeared to specify validation rules. E.g., Check language [55] from openArchitectureWare pretends to simplify the definition of the validation rules.

For model verification there is an approach based on reducing the problem to the Constraint Satisfiability Problem using constraint programming [88].

Using model comparison technique there is a framework able to verify model transformations [89-91]. As said before, the problem of this technique is that many times we get false positives.

Finally, unit testing is a technique that has been also applied to MDE [92] combining methods from transformation testing and constraint checking, in this case a specific language based on Epsilon languages [93] is used to define tests.

In Table 10 a comparison of the constraint languages that appear in this section is shown.

	Standard compliance	Principal benefit	Principal disadvantage	Multiple models	Facilities	Language
Check	No	Specific language	Depends on oAW	Unknown	Integrated in oAW	Declarative
Epsilon	No	Specific language	Not mature enough	Yes	Automatizable tests	Declarative
OCL	Yes	Expressi-veness	Complexity and error-prone	No	No	Declarative

Table 10: Comparison between constraint languages

4. Non-Functional Requirements in MDE

To explain why the Non-Functional Requirements (NFRs) are important in MDE, first we must define the term. There are many definitions for the term NFR discussed in [12]. From all the definitions exposed in that work, the one that better applies for MDE, in my opinion, is the definition of I. Jacobson, G. Booch, and J. Rumbaugh [94]:

"[NFR is...] a requirement that specifies system properties, such as environmental and implementation constraints, performance, platform dependencies, maintainability, extensibility, and reliability. [NFR is...] a requirement that specifies physical constraints on a functional requirement."

The majority of software development life cycles have at least one step to treat with NFRs. During the software development, it is primordial to take care of NFRs and Software Quality (SQ) aspects. L. Chung et al. [95] argued that the lack of integration of NFRs with functional requirements, can result in long time-to-market and more expensive projects (this fact is also mentioned in the famous "No Silver Bullet" article of F. P. Brooks [96]).

NFRs in MDE will bring the possibility to reason with them during the engineering processes in a semi-automatic way. MDE offers the possibility to include NFRs as part of the engineering processes as something more relevant than documentation. MDE allows representing these NFRs by modeling them and/or by setting them as a part of the driver mechanism of the development process.

Studying the related works to NFRs in MDE research topic, I have found, that there are different (but not necessarily exclusive) approaches to treat with NFRs in MDE, basically there are two families of works.

- Notation of NFRs in models:
  - Many works are based on UML profiles, in concrete the OMG Modeling and Analysis of Real-Time and Embedded systems (MARTE) profile [97], the paper [98] explain the notation part of this profile.
  - Specific modeling languages for NFRs, examples are [95, 99-101].
- Using NFRs in MDE:
  - Selecting the transformations depending on the expected quality, examples of this can be found in [102-104].
  - Mechanisms centered in validation of the NFRs are also an alternative, a work in this direction is presented in [105, 106].

In the next subsections these approaches are explained in more detail.

4.1. Notation of NFRs in models

For the notation of NFRs in models there are two perspectives. One is to extend the modeling language to support NFRs representation, and the second is the use of well known modeling languages thought to represent NFRs.

In the first perspective, the extended modeling language is normally UML because, nowadays, UML is the most used language for software specification and design. Many research efforts are centered on extending this language with UML profiles.

On the other hand, DSLs aim at using small, well-focused language that provides appropriate abstractions and notations for a particular domain.

In the last years it seems to be a tendency to use DSLs instead of UML profiles, but it is not clear which modeling technique is better. Microsoft defends DSLs while OMG have done many standardization efforts for UML profiles.

4.1.1. UML Profiles

UML profiles allow the customization and extension of the modeling language syntax and semantics to define specialized modeling languages for particular domains.

For NFRs, MARTE [97] is the most known UML profile. MARTE can be extended, for example, in [107], D. C. Petriu et al. present an extension to give support for dependability analysis. Another profile that explores dependability property is DMP [108]. Also in the real-time systems modeling area, other profiles (e.g. UML-SPT [109], UML-RT [110]) take into account non-functional properties like performance.

The approach presented in [111] uses a UML profile, named UP-SNFR, to support the specification of NFRs inside UML models for SOA systems. It is based on the idea of features.

The UML profiles used in [112] are for specifying architectural decision and NFRs, the good thing is that they can be used together. This is important because as stated in the same paper "*The rationale behind each architecture decision is mostly about achieving certain NFRs*".

Other interesting profiles that take into account some form of non-functional property are:

- UML profile for Quality of Service (QoS), QoS-FT [113]
- UML profile for Security, UMLsec [114]

4.1.2. GORE specific languages

Goal-Oriented Requirements Engineering (GORE) is an approach advocating for the identification and analysis of goals as a prerequisite for writing of complete and consistent requirements documents. Goal models are commonly used in GORE.

In the previous sections I have explored two works that use goal models:

- In [46] goal models are used, but no mention is done for NFRs.
- In [73] goal models contemplate the existence of NFRs, but it is not clear how they are used inside the proposed methodology.

E. Yu is recognized as the author of the most used goal modeling language, *i* [115]*. This modeling language has many similarities with NFR as explained in [116]. In [99], E. Yu has exposed his thoughts about the directions to take, and the open problems to support NFRs with goal modeling in MDE.

More recently, L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos have defined the NFR Framework in [95]. This framework describes a notation to specify NFRs based on goals and soft-goals, and it is accepted by the requirements engineering community. It has many common elements with *i** language. L. M. Cysneiros have several works [100, 101] to use this notation inside conceptual models.

4.2. Use of NFRs in MDE

In the studied works there are two usages for NFRs inside MDE. One use the NFRs to drive the transformations and the other uses NFRs to validate the results of the transformations.

4.2.1. NFR-driven Transformations

When we say that NFRs play the driver role in transformations, we are saying that the selection or the behavior of the transformations will depend on the specified NFRs.

Some works in this direction are presented as quality-driven transformations. The most known quality standard is ISO 9126 [13], currently it is being substituted by ISO/IEC 25000 [117]. In [117] the software quality is defined as "the capability of software product to satisfy stated and implied needs when used under specified conditions".

In the beginning of this Master Thesis it was said that software quality is composed by functional and non-functional factors. This is specially true when we look at quality-driven transformations, like the one presented in [102] to tackle with usability, where only the functional part of the usability is used to drive the transformations. The same happens in [103], it is not clear if they take into account the non-functional factors for the understandability aspects.

An earlier work [104], was proposing similar things in the context of CBSE. This work uses measuring mechanisms of the Quality of Service (QoS) to transform the system.

4.2.2. Validation of NFRs

Instead of using NFRs to drive the transformations during the development, there is one approach, presented in [105, 106], that uses them to validate the resulting product. This approach proposes a separation of NFRs in different levels to do the validation in each stage of the development process. In a latter publication [118] based on the same work it is also considered a run-time stage for validation.

For me, the most interesting thing of this approach is the differentiation of NFRs in the three kinds of models: computational independent, platform independent and platform specific.

4.3. Comparison of approaches

In Table 11 and Table 12 shown the comparison between the approaches presented in section 4.

	NFRs representation	Supported NFRs	Architectural constraints	Effects over MDE	Automation
GORE	Specialized Models and languages	Any	Recognized as an open problem	Need support for goal models	Recognized as an open problem
UML profiles	UML	Depends on the used profile	Depends on the used profile	Independent	Need support for the extension

Table 11: Comparison between NFRs notation

	NFRs representation	Supported NFRs	Effects over MDD	Generated artifact	Automation	Validation
Transformations approach	Quality metrics	Usability Understandability	Featured transformations	Models/Code	Seems feasible	Unknown
Validation approach	Specialized Models	Performance Reliability	New models to be considered	Unknown	Not applicable	During the development and execution time

Table 12: Comparison between NFRs usage approaches

The notations and approaches to use NFRs are not being accepted by the software development community. Some of these UML profiles are currently used, but mainly in the cases where the model acts as documentation.

4.4. Principal research groups of NFRs in MDE

Several research groups have been identified in the exploration of this research topic:

- D.C Petriu (Canada), E. Yu (Canada), and L. M. Cysneiros (Brazil): these three groups have special interest in NFR notation.
- S. Abrahão (Spain) and O. Pastor (Spain): These two groups have special interest in quality-driven transformations.
- P. Inverardi (Italy) and C. Ghezzi (Italy): These two groups have special interest in validation of NFRs.

5. Analysis of the systematic review

This section contains an analysis of the reviewed works to ensure that the results exposed in this systematic review are significative enough. Basically there are two kinds of analysis, one is to summarize the relevance of each studied work (see Table 13), and the other one is to summarize the amount of work studied of each topic (see Table 14).

5.1. Analysis of quality

In Table 13 the summary of the documents examined for each topic is represented. Columns represent (in order): Reference, the type of information source, the year of publication, and the number of citations.

Considerations for the qualitative analysis:

- The number of cites exposed in Table 13 were obtained in Google Scholar in May, 2009.
- Some of the studied works for this systematic review are too recent to be evaluated by its citations. The most recent articles are pre-print versions.

	Ref.	Src.[2]	Year	Cites
MDE generic	**[4]**	**A**	**2006**	**286**
	[16]	**B**	**2004**	**246**
	[18]	**B**	**2002**	**496**
	[19]	**B**	**2003**	**567**
	[20]	**B**	**2003**	**809**
	[29]	**A**	**2003**	**281**
	[31]	**B**	**2003**	**204**
	[34]	**A**	**2005**	**205**
	[36]	**A**	**2003**	**86**
	[38]	**C**	**2005**	**25**
	[39]	**C**	**2005**	**6**
	[40]	**C**	**2003**	**148**
	[41]	**A**	**2006**	**4**
	[42]	**A**	**2003**	**208**
	[45]	**C**	**2005**	**1**
	[58]	**A**	**2006**	**55**
Domain-Specific Modeling	**[69]**	**C**	**2007**	**5**
	[70]	**C**	**2007**	**0**
	[71]	**C**	**2007**	**10**
Methodologies	**[17]**	**B**	**2007**	**30**
	[72]	**C**	**2008**	**0**
	[74]	**C**	**2006**	**2**

[2] A: Journal article, B: Book, C: Conference paper

	Ref.	Src.[2]	Year	Cites
	[73]	**C**	**2007**	**2**
Transformation languages	**[46]**	**C**	**2007**	**8**
	[52]	**A**	**2006**	**117**
	[56]	**C**	**2006**	**1**
	[75]	**C**	**2007**	**0**
	[76]	**C**	**2006**	**0**
	[77]	**C**	**2006**	**2**
	[78]	**C**	**2006**	**19**
	[79]	**C**	**2007**	**8**
	[80]	**C**	**2005**	**120**
	[81]	**A**	**2009**	**2**
Correctness	**[85]**	**C**	**2006**	**3**
	[83]	**C**	**2006**	**35**
	[86]	**C**	**2006**	**8**
	[87]	**C**	**2007**	**3**
	[88]	**C**	**2008**	**3**
	[89]	**C**	**2004**	**20**
	[90]	**B**	**2005**	**30**
	[91]	**C**	**2006**	**10**
	[92]	**C**	**2008**	**2**
Non-Functional Requirements	**[5]**	**A**	**2005**	**32**
	[12]	**C**	**2007**	**15**
	[94]	**B**	**1999**	**174**
	[95]	**B**	**2000**	**1026**
	[98]	**C**	**2005**	**19**
	[99]	**C**	**2008**	**1**
	[100]	**C**	**2001**	**31**
	[101]	**A**	**2004**	**75**
	[102]	**C**	**2008**	**0**
	[103]	**C**	**2008**	**0**
	[104]	**C**	**2004**	**22**
	[105]	**C**	**2007**	**5**
	[106]	**C**	**2007**	**6**
	[107]	**C**	**2008**	**1**
	[108]	**C**	**2003**	**3**
	[111]	**C**	**2007**	**2**

	Ref.	Src.[2]	Year	Cites
	[112]	**C**	**2007**	**9**
	[115]	**C**	**1997**	**458**
	[116]	**A**	**2002**	**363**
	[118]	**C**	**2008**	**0**

Table 13: Qualitative analysis

Some of the reviewed papers have few or zero citations, this situation could be explained by many factors:

- As explained in the search strategy (section 1.3), I did not use the number of citations to select the primary information sources, instead of that I used an exploratory search strategy focused on the principal conferences and journals of the research area.
- Google Scholar does not have all the knowledge and many citations could be lost. This case is more relevant for the non-English papers.
- As said before, some referenced works are too recent to be evaluated by its citations.

Many researchers do not think that the number of citations represents the quality of the paper, but it is hardly related with the community acceptance.

5.2. Quantitative analysis

In Table 14 the summary of the number of documents examined for each topic and its type is represented.

	Journal articles	**Books**	**Conferences / Workshops**	**Electronic sources**	**Total**
MDE generic	**7**	**5**	**4**	**13**	**29**
Frameworks and CASE tools	**0**	**0**	**0**	**10**	**10**
Domain-Specific Modeling	**0**	**0**	**3**	**1**	**4**
Methodologies	**0**	**1**	**3**	**0**	**4**
Transformation languages	**2**	**0**	**8**	**7**	**17**
Correctness	**0**	**1**	**8**	**2**	**11**
Non-Functional Requirements	**3**	**2**	**15**	**5**	**25**
Total	**12**	**9**	**41**	**38**	**100**

Table 14: Quantitative analysis

In difference with the analysis of quality, in the quantitative analysis electronic sources have been considered for quantification.

MDE topic in general is covered from all types of sources, but books are the most important source in this case.

Frameworks and CASE tools, not-being a research topic it is normal to see that all the information sources are electronic sources (web pages of the software products).

It is noticeable that the Domain-Specific Modeling and Methodologies do not have many examined documents but, in my opinion, the topics are not crucial for the NFRs in MDE topic and have been sufficiently covered.

Finally, the NFRs topic has been explored more accurately. Proof of that is the number of conference papers examined. They are almost the half of the total.

Figure 10 shows the number of examined documents and type of each year. It is noticeable that I focused my search in the 2006-2008 conferences. It is also important to notice that the most important books of the research area were published during 2003-2004.

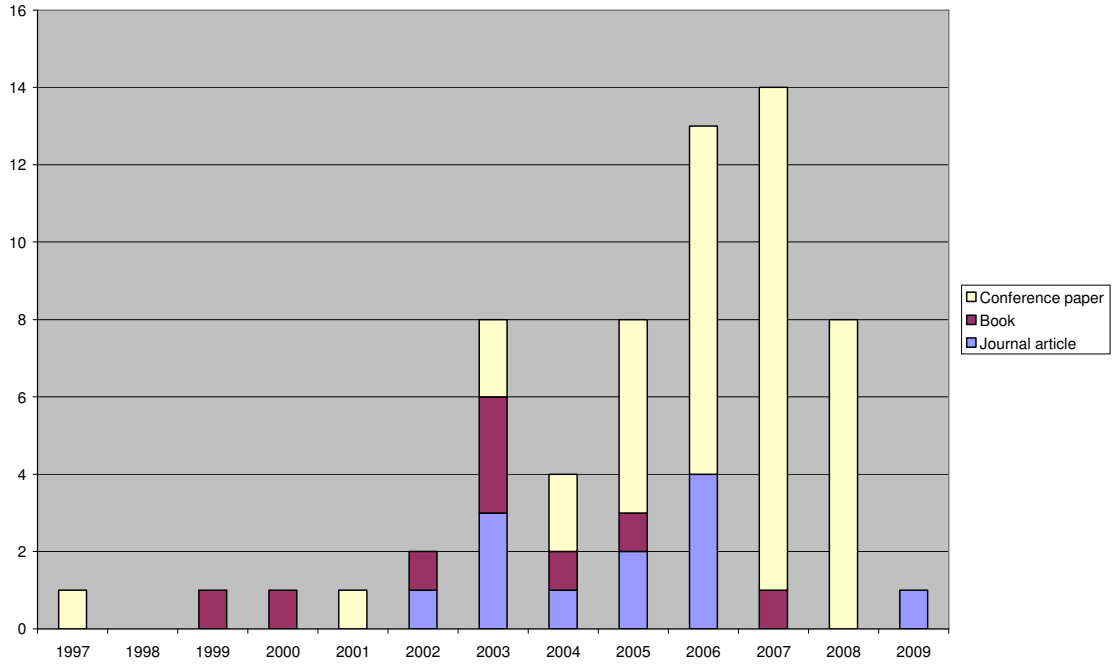

Figure 10: Number of documents/year

6. Conclusions of the systematic review

In this systematic review the following goals have been accomplished:

- Definition of a protocol for the systematic review.
- Definition of a research question based on a justified hypothesis.
- Historical exploration of the research done around the studied research area.
- Overview of the current and common knowledge of the studied research area.
- Review of main research topics associated with the research area.
- Particular review of the research topic where my work is focused.
- Validation of the systematic review.

The definition of the protocol is based on the indications made by B. Kitchenham in [15]. I have established a rigorous protocol to reduce a possible bias in the results. The protocol contains all the strategies used to search and to extract data.

I have justified my research question, basing my reasoning on historical facts and parallelisms with other research areas related with MDE.

I have presented a full overview of the MDE research area, based, principally, on the most cited books of the area. In particular, I wrote this part of the document as an introduction. A particular emphasis has been made on all the new terminology introduced by this research area. As a result, I made the entire document self-understandable, even for a non-expert in the field.

For a further exploration of this research area, I have selected the research topics that are present on the majority of conferences that are specific to MDE. For each of them I did a particular search for the relevant contributions done in the last years.

To complete the systematic review, I made a specific section for my current research topic, the support of NFRs inside MDE. For this section of the document I have done a deeper search looking for older contributions and following many of the references of published works.

Finally, the validation of this systematic review has been done taking into account the amount of reviewed work for each topic and the importance of each studied work.

6.1. Answer to the research question

In the beginning of the systematic review I stated the following question:

> *Are Non-Functional Requirements correctly supported by the current state of Model-Driven Engineering?*

As said at the beginning of the Master Thesis, the NFRs will be correctly supported in MDE when:

- All kinds of NFRs should be specifiable, preferably in a commonly accepted notation.
- The produced software with a MDE process should be compliant with NFRs specifications.

The first part of the "correctly supported" definition is not really accomplished. Some of the studied works claim generic support for NFRs, but in practice only a few NFRs are supported in each work. On the other hand, many of the studied works in this topic used its own notation to specify NFRs. The common accepted notation for representing NFRs or requirements in general, in GORE, is the NFR Framework [95], and the *i** modeling language [115]. Only a little amount of works in MDE are using this notation.

The second part of the definition can be tackled in two ways, as an empirical validation of the software product (e.g., using some testing technique) or with a formal proof to state that the development process will obtain a software product compliant with the specified NFRs. Normally, in engineering areas the first way is more common. In this systematic review I have referred to some works that used validation for NFRs (section 4.2.2), but they mainly focuses on the ideas, not in how the NFRs are validated.

My answer to the research question is that, after getting a full background on MDE, and reading all the works I found about NFRs in MDE, I am convinced that MDE research area needs much more effort to correctly support NFRs. These are the research topics related with NFRs and MDE that require more attention:

- Notation of NFRs in models
- NFR-driven transformations (determine the role of NFRs)
- Case of study of MDD using NFRs
- Empirical validation of MDD using NFRs

To get a more complete answer to this question with the view of the non-academic world, in the third part of this document an ongoing survey for IT companies and organizations is presented.

6.2. Final note

Software engineers have a long term goal to achieve; the introduction of the MDE ideas in software industry. Whenever this goal is accomplished, it should be with NFRs support.

Currently, although requirements engineering has many years of experience, it seems that the support of NFRs inside MDE is in its first steps. A proof of this fact is that the researches are exploring different and new approaches to treat this problem, and there is no consensus on the role and the use that NFRs should play inside MDE.

In this situation, create novel work seems to be feasible. Also, in this situation, it is easy to fall into the error of trying to start works from scratch. As a learned lesson, the intention of my work is to reuse and take ideas from the current ongoing works in this research topic and from the major related research areas (e.g., requirements engineering, software quality, and software development methods). Some of these ideas have been already retrieved doing this systematic review, but the reviewing work has to be maintained as long as the research remains.

Part II

Responsibility Detection and Transformation (RDT):

Framework architecture

7. Introduction to RDT framework

This part of the Master Thesis contains a referential architecture of Responsibility Detection and Transformation (RDT) framework.

This framework is an evolution of a previous work done for my final career project [7]. For the final career project a proof of concept, AR3L, was developed. It is available in [11].

RDT has been previously published in a conference paper [8], in a workshop paper [9], and in an internal report [10], in each publication different aspects of the framework where considered. What is presented in this Master Thesis is a summary (self contained) of the advances done in the RDT framework during the last year.

RDT is a framework which its principal objective is developing software using the MDE ideas. RDT help the designer to evolve the specification into the design (see Figure 11). This is a concrete step inside the classical software development cycle of life.

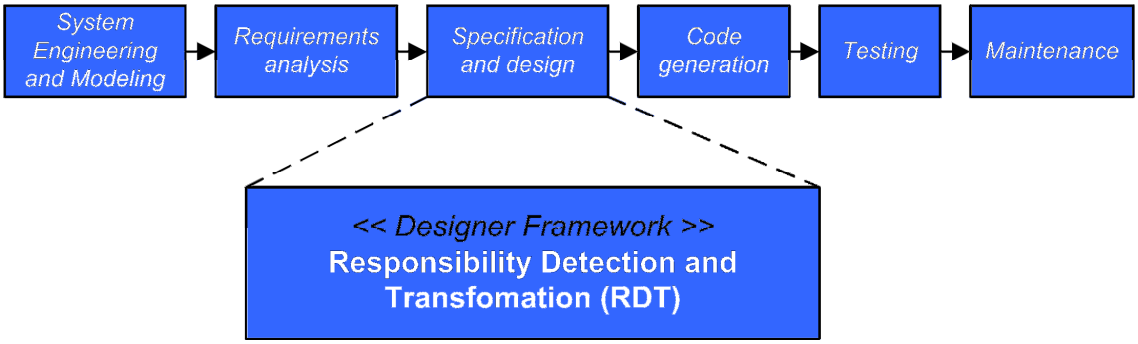

Figure 11: Role of RDT in the classical software development life cycle

Together with the principal RDT objective the following aims are taken into account:

- A special emphasis is done for taking into account NFRs during the software development. As seen in the conclusion of the systematic review it is necessary to improve the support of NFRs in MDE. RDT tries to fill this gap in the research of MDE.
- Help the designer without replacing him. This framework should contemplate several interaction steps that allow the designer to make decisions and eventually shiftily modify the intermediate results.
- Be adaptable, the behavior of RDT shall be highly customized.
- Be open, all the components of the architecture shall be exchanged with existing ones.
- Improve the reusability and portability of the software product by applying the Model-Driven Engineering (MDE) ideas.

7.1. Classical software development life cycle

As seen in Figure 11, RDT is thought to be a part of a large development life cycle, the classical software development live cycle (a.k.a. waterfall life cycle). This life cycle has been used for many years, and almost all the current software development life cycles are based on the same steps but

correcting or improving some of the detected lacks (e.g. iterative life cycles).

The classical software development life cycle has been chosen because it is well known by the software engineering community. Each step is documented in many popular books of software engineering (e.g. [119]). With this documentation and common knowledge it is a feasible task to specify the necessary components and processes to go from specification to design models.

Without any concrete considerations, RDT will be easily adaptable to any other life cycle that considers the step of going from specification to design models.

7.2. Model-Driven Development

There are several differences (see Figure 12) between RDT and Model-Driven Development (MDD):

- The first difference is that RDT does not consider CIM. This model has its equivalence on the two initial steps of the classical software development life cycle. Currently these steps are not considered inside RDT, however part of the information needed by RDT is obtained from the requirements analysis step. The non-functional requirements.
- The second difference is that in the MDD context Platform Independent Models (PIM) are limited to models that are independent of the platform. Specification models are normally independent of the platform but not necessarily. The definition of a specification model used by RDT is: A specification model is any model from which *responsibilities*[3] can be inferred, in this situation almost any model can be a specification model [119].

  A benefit obtained from this fact is that when we talk about obtaining a PIM from a Platform Specific Model (PSM), in MDD it is reverse engineering while in RDT the PSM could be a different kind of specification model.
- The third difference is about the number of models. In MDD we normally work with one model that is transformed to one model. RDT contemplates the possibility to work with multiple heterogeneous[4] specification models and obtain multiple heterogeneous design models.

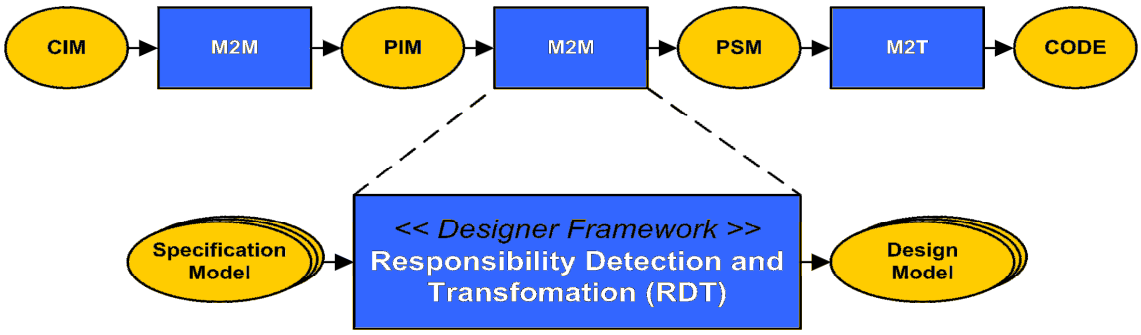

Figure 12: Comparison of RDT with MDD

[3] The term *responsibility* is defined in the glossary.

[4] By *heterogeneous* we mean that they are from different metamodels.

These differences (in particular the second one) are important enough to say that RDT is not compliant with what MDD have established, but it is true that the principal ideas are the same.

8. RDT Architecture

RDT architecture follows the pipe and filter [120] architectural style (see Figure 13). Following the flow expressed in the diagram, every RDT process generates a model that is one step nearer to the design models.

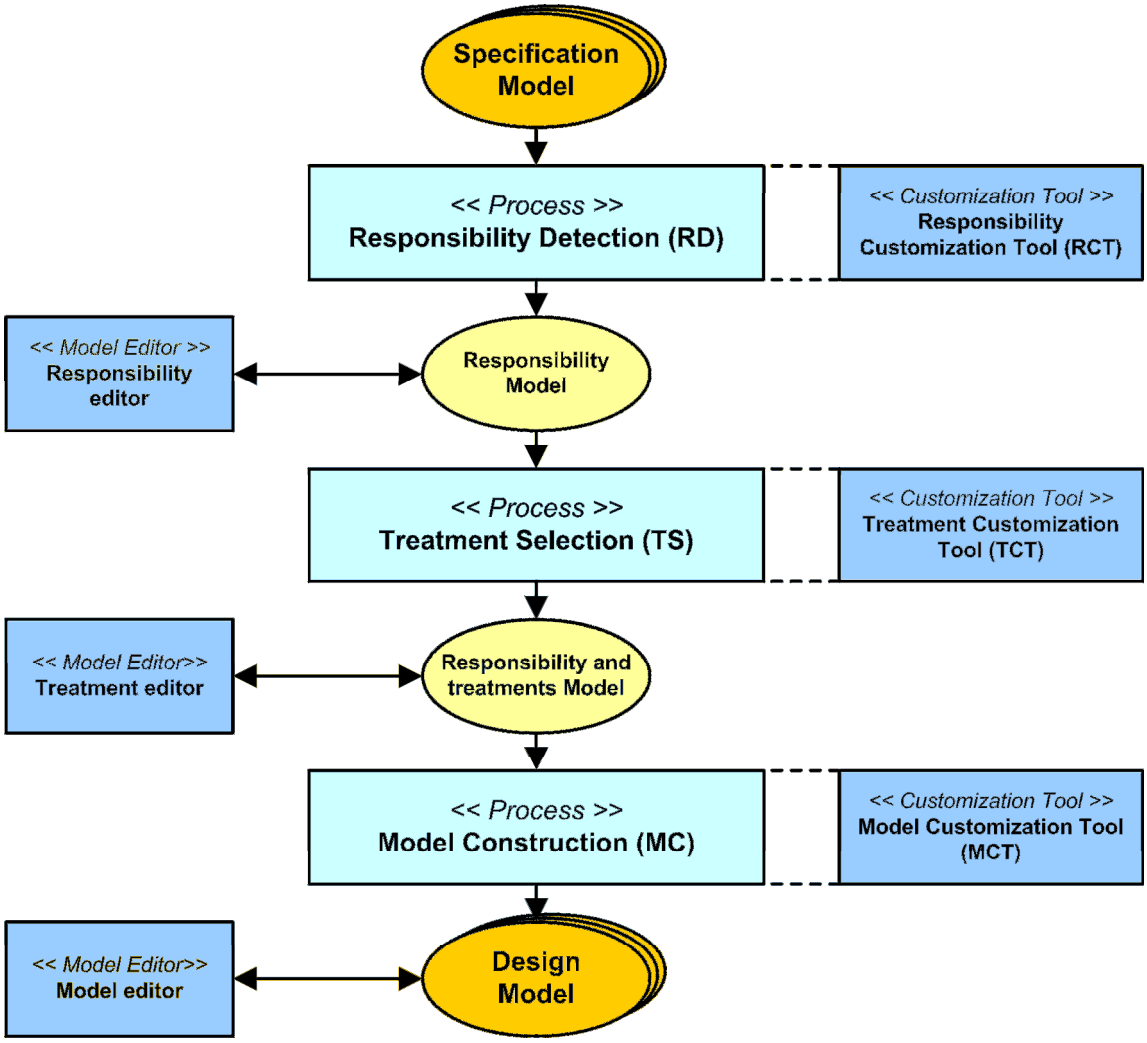

Figure 13: RDT architecture

The core of RDT is the concatenation of the RD, TS, and MC processes (section 8.1). In order to have a customizable architecture the RDT behavior of these processes are stored in repositories (section 8.2). These repositories can be managed with the customization tools, each one associated with one process. It is previewed an interaction step in each intermediate stage in which the user can adapt the generated model to his or her needs with the model editors (Section 8.3).

For better understanding how this framework works, there is an example in the AR3L web page. This example is based on the previous version of this framework but it is still valid to get an idea of what can be done with this framework. The Model Construction was not considered in the AR3L tool, so the final artifact is a list of responsibilities with the applicable treatments. The URL of the example is: **http://www.lsi.upc.edu/~gessi/AR3L/samples.html**

8.1. RDT core

This section is focused on understanding the architectural aspects of each step of the RDT processes: RD, TS, and MC.

Two notations are used to explain each part of the architecture (see Figure 14 and Figure 15). One is the own-made notation that is easy to understand, and the other is UML notation. The acronyms used in the UML figures are equivalent to those shown in own-made notation but in the UML figures represent components while in the own-made notation represent processes.

Figure 14: RDT core

RD uses the specification models to generate one responsibility model. TD adds the treatments information in the responsibility model. Once the responsibility model is completed it is used by RT tool to generate the design models.

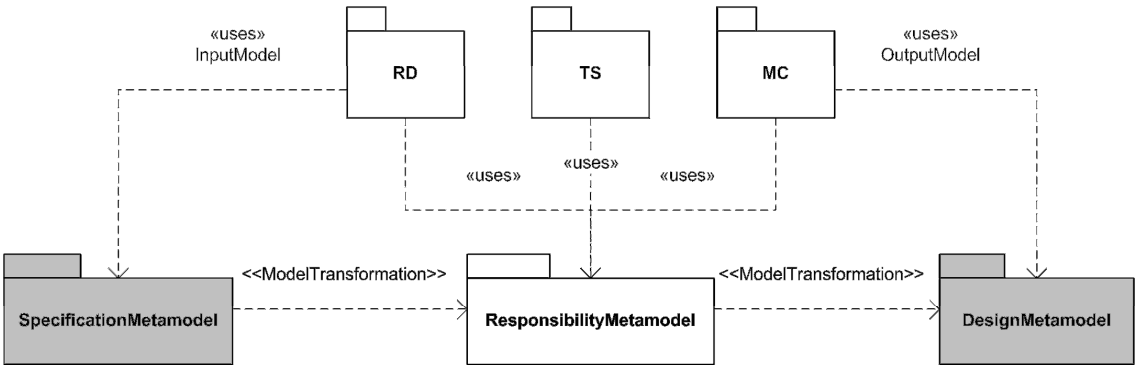

Figure 15: RDT core (UML)

Instances of *Specification Metamodel* and *Design Metamodel* could apply for the same metamodel or for different ones. They are differentiated in the UML model to show the model transformation arrows. It is also important to notice that these two metamodels are used by RDT, but could be defined by the user. This is interesting when RDT framework is used with DSLs.

In the next sections the core processes are detailed.

8.1.1. Responsibility Detection

Figure 16 shows the architectural view for this process. This process is divided in two: detection of responsibilities from the specification models and unification of the detected responsibilities in one responsibility model.

We can have any number of *Responsibility Detection* processes, each one with one specification model from the same or different metamodels than the other processes. *Responsibility Detection* process uses the *Responsibility repository* to know the types of detectable responsibilities.

Note that there could be two models from the same metamodel, but using different mechanisms of specification, for example two UML class diagrams, one specifying the identifiers with OCL, and one using a particular stereotype. In this situation there will be more than one mechanism to detect one kind of responsibility for one kind of metamodel.

The *Responsibility Unification* process is unique. It gets all the detected responsibilities, and generates one *Responsibility model* that is an instance of our *Responsibility metamodel*.

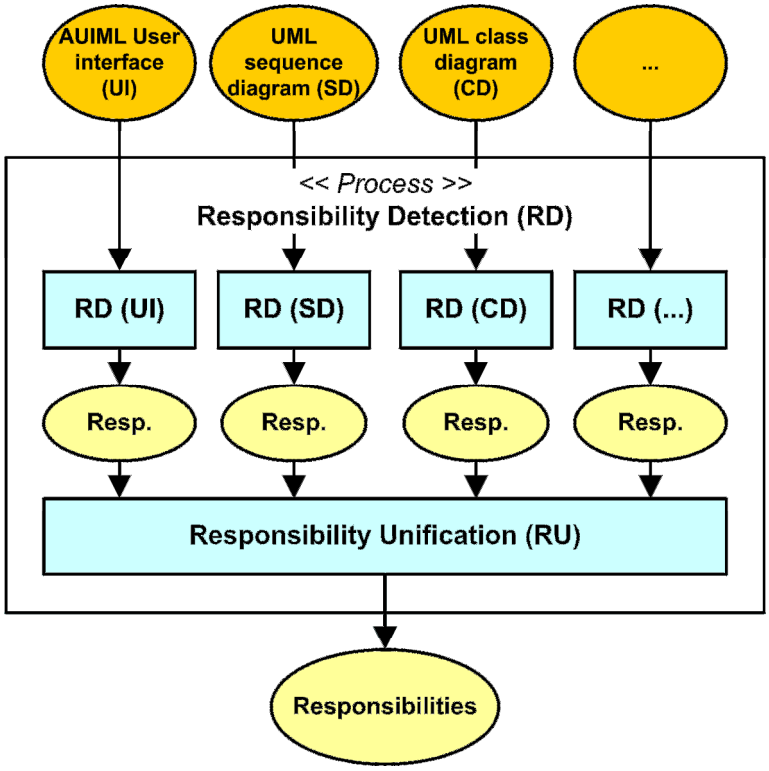

Figure 16: Responsibility Detection

As an example, in Figure 16 there are three specification models: a user interface described in AUIML, a sequence diagram described in UML, and a class diagram also in UML.

The UML view of RD is shown in Figure 17. Each instance of *Responsibility Detector* is responsible of executing a *Responsibility Detection* process, and the *Responsibility Unifier* is responsible of executing the *Responsibility Unification* process.

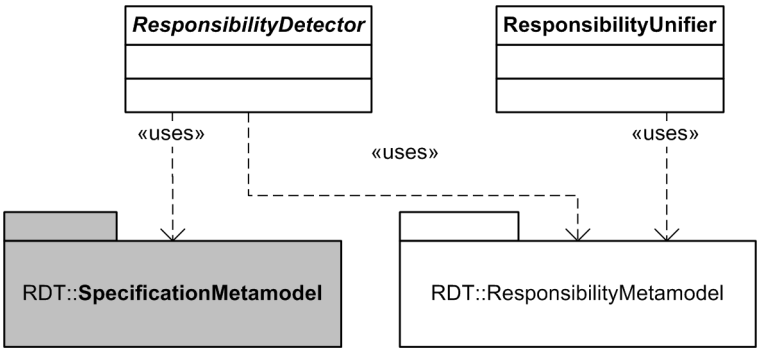

Figure 17: Responsibility Detection (UML)

8.1.1.1. Responsibility Detector

The *Responsibility Detector* needs the detection mechanisms associated with the *Specification Metamodel* to work with the specification model. Note

that the detection of a responsibility could be different from one metamodel to another or it could be not detectable in some metamodels.

The detection mechanism for a concrete responsibility is associated with a specification metamodel and is defined in the *responsibility repository.* Using RCT (explained in section 8.2.2) the user will be able to add or modify the way of detection for each responsibility and for each kind of metamodel. The detection mechanisms will be defined in a logic-based language (similar to OCL).

It is necessary to preprocess the input model (in this case the specification model) to validate it because there are many ways to detect a responsibility, and a responsibility can be expressed in different ways. *Model Validator* (see Figure 18) is used to do this task. This component uses the constraints associated with in the *Specification Metamodel* to verify that input model is valid. The user is able to extend the *Specification Metamodel* with new constraints. These constraints are stored in the *Responsibility repository,* and managed by RCT.

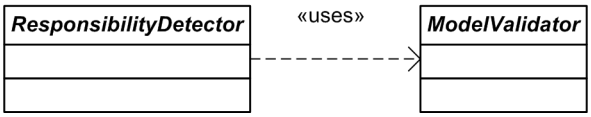

Figure 18: Model Validator

Once the detection is finished the *Responsibility Detector* generates an instance of the *Responsibility Metamodel* with the detected responsibilities.

8.1.1.2. Responsibility Unifier component

This component does the operation described in Equation 1, where *y* is an input model, and *RD* is the *Responsibility Detector* component. *X* represents an instance of *Responsibility Metamodel* with all the detected responsibilities.

$$X = \bigcup_{\forall y} RD(y)$$

Equation 1: Responsibility Unifier

To ensure the correctness the *Responsibility Unifier* does integrity validation over the unified responsibility model. This validation will try to detect and repair duplicate responsibilities. Naming convention seems to be the simplest way to face this problem.

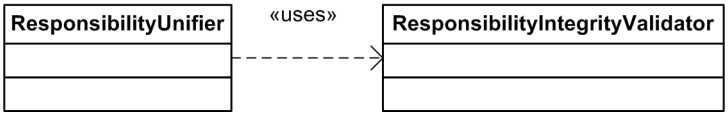

Figure 19: Responsibility Integrity Validator

8.1.2. Treatment selection

Figure 20 shows the architectural view for this process. This process will use the detected responsibilities and the NFRs. It is also considered the option that the user pre-establishes some of the treatments (e.g. the user wants to use stored procedures as a constraint for the whole system).

The process for selecting the best treatments is divided in three parts:

- *Architectural Treatments Selection (ATS):* This step selects the treatments that bring together responsibilities to compose architectural components. The selection of these treatments depends on the best matching architectural style for the NFRs.
- *Design Treatments Selections (DTS):* This step selects the treatments that bring together responsibilities referring to a primary element (entities). Also in this case NFRs will guide the selection of treatments.
- *Technological Treatments Selections (TTS):* In this step the identified the components and the primary elements are used to select the technological treatments. Also in this case NFRs will guide the selection of treatments.

When the three kinds of treatments are selected we have all responsibilities with at least one treatment designated.

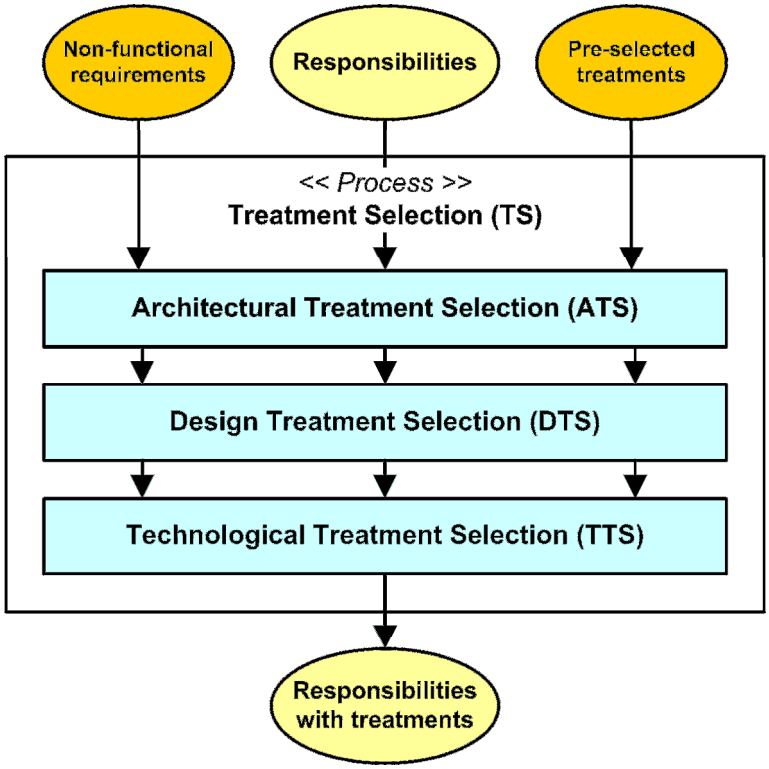

Figure 20: Treatment Selection

In the UML representation (Figure 21) we can see that the Treatment Selector (TS) component can be reused in each of the three steps described adobe.

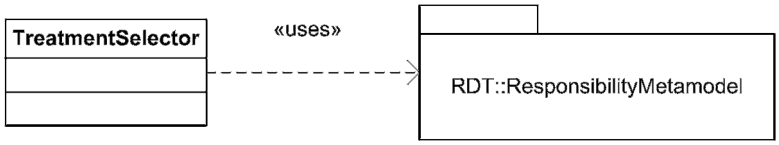

Figure 21: Treatment Selection (UML)

8.1.2.1. Treatment Selector component

This component will select the treatments considering the specified NFRs and the user pre-selections of treatments. This component is thought to use

a maximizing algorithm that tries to find the best combination of treatments to construct the software system (other implementations are possible).

For each treatment we will have the information of the favored or disfavored NFRs by the application of the treatment.

8.1.3. Model Construction

When the more adequate treatments for the responsibilities are selected we have enough information to construct the software product. The Model Construction process is responsible to do this job.

As said at the beginning of the documentation of this framework, RDT is capable to generate one or more design models. In Figure 22 we can see how this process can generate several heterogeneous models from the same set of responsibilities and treatments (e.g. the UML design model will contemplate several diagrams like class, use case and sequence diagrams). It is important to notice that not all responsibilities can be represented in all kinds of models. For each model, the responsibilities that it is capable to represent will be included.

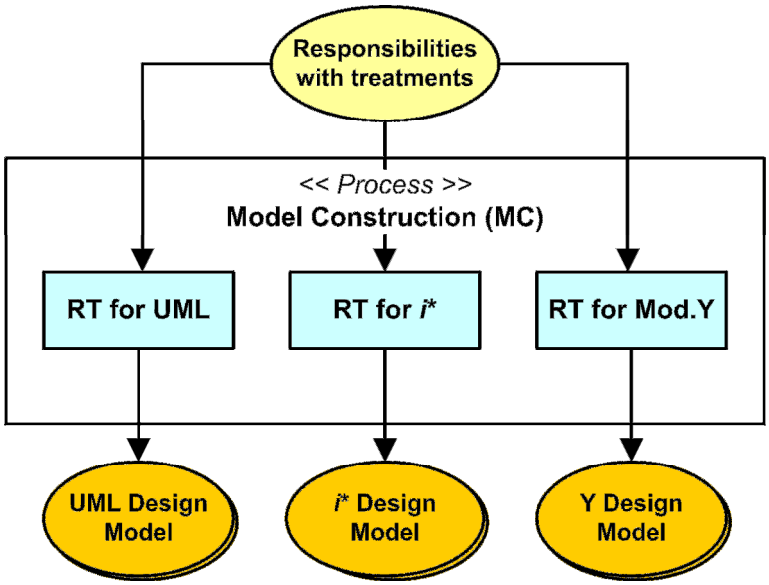

Figure 22: Model Construction

In the UML representation (see Figure 23) the Responsibility Transformer (RT) component is shown. As before the same component is used for every type of transformation.

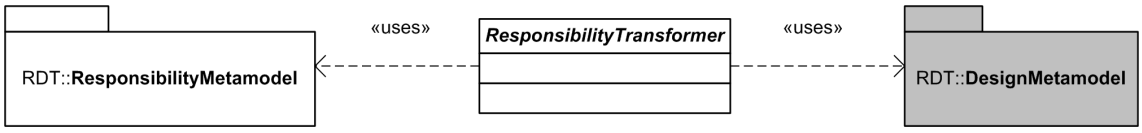

Figure 23: Model Construction (UML)

8.1.3.1. Responsibility Transformer component

A treatment is some kind of abstract transformation, a transformation that has a semantic use but is not applicable because it misses the mapping with the design metamodel. In other words, each treatment will be associated to several mappings from responsibilities to a concrete design metamodel.

The responsibility transformer will apply the specific mapping for each of the selected treatments to obtain the concrete design model.

8.2. RDT customization

As seen in the previous sections, many components can be reused in many places. This fact can be explained because the specific behavior is stored as data in several repositories. Storing the behavior of the components has many benefits. One of them is the possibility to customize the behavior.

8.2.1. Repositories

There are two kinds of repositories ones that are thought to contain the static elements (like NFRs) that intervene in the processes, and ones that contain the behavior between elements (like treatments).

These repositories can have two orientations: one is to use them locally and the other is to use them oriented to community. In the second case many of the efforts of building transformations for specific design models can be reused by all the users. Also it is possible to assign roles to these repositories (e.g. expert users will be able to introduce new stuff while normal users will only be able to read the stored data).

In Figure 24 we can see the main concepts and relations. They are explained in more detail in each repository.

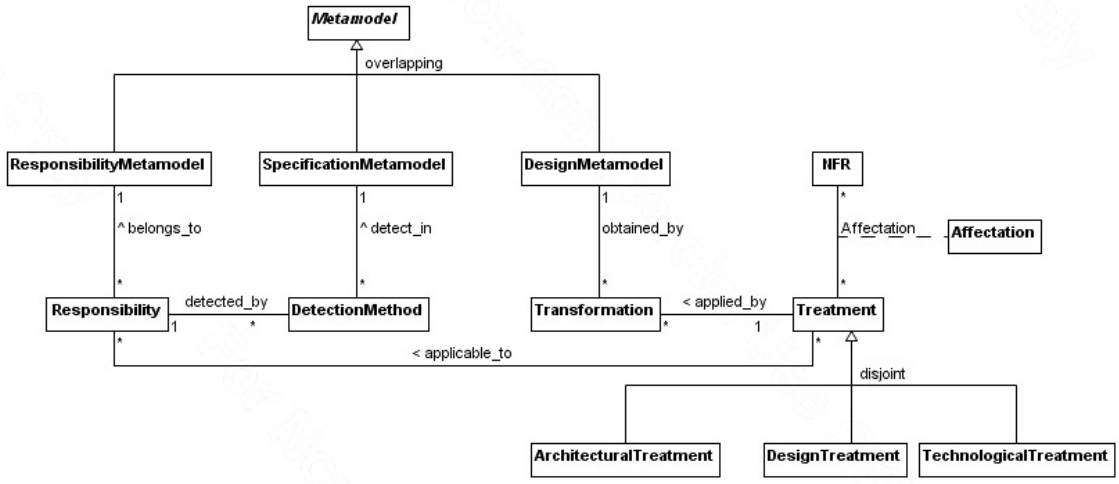

Figure 24: Main concepts and relations

The next subsections will describe the main repositories of this framework.

8.2.1.1. Metamodel repositories

Metamodels repository will contain all necessary metamodels. Three particular kinds of metamodels are used in this framework:

- Responsibilities metamodel: This metamodel is unique and is used all along the processes described in this framework.
- Specification metamodels: These metamodels are necessary to detect responsibilities. The responsibilities are associated through the detection methods with these metamodels.

- Design metamodels: These metamodels are necessary to construct the design model and to apply the selected treatments. The treatments are associated through the transformations with these metamodels.

8.2.1.2. Responsibilities repository

Responsibilities repository will contain a static base of responsibilities that are susceptible to be detected in specification models. For each specification metamodel the repository can have one or more detection method.

8.2.1.3. Treatments repository

Treatments repository will contain a static base of NFRs that are associated with the treatments. The treatments are classified in three kinds as explained in section 8.1.2.

8.2.2. Responsibility Customization Tool

Responsibility Customization Tool (RCT) is the tool that will allow to the user of this framework to customize the detection methods for each responsibility.

In the beginning the set of responsibilities will be static, but in the future this tool will also allow including new responsibilities.

This tool will also allow the addition of net detection methods for a specific specification metamodel.

8.2.3. Treatment Customization Tool

Treatment Customization Tool (TCT) is the tool that will allow the user of this framework to customize the levels of affectation between a NFR and a treatment.

In the beginning the set of NFRs will be static, but in the future this tool will also allow including new NFRs.

This tool will also allow the addition of new architectural treatments, design treatments, and technological treatments. When a new treatment is created it has to be associated with one or more responsibilities where the treatment is applicable.

8.2.4. Model Customization Tool

Model Customization Tool (MCT), is the tool that will allow the user of this framework to customize and create new transformations for a specific treatment and design metamodel.

8.3. RDT interaction steps

As said at the beginning of the documentation of this framework, It is previewed an interaction step in each intermediate stage in which the user can adapt the generated model to his or her needs with the model editors.

8.3.1. Responsibility Editor Tool

Responsibility Editor Tool (RET), this tool will allow the user of this framework to modify the detection of responsibilities. In some cases it will be very complicated to define a good detection method that contemplates

all possible cases. For this reason and because many times we do not model everything this tool will allow to complete the set of responsibilities that are needed for a software system.

8.3.2. Treatment Editor Tool

Treatment Editor Tool (TET), once the responsibilities are detected, this tool will allow to change the selection of treatments done taking in consideration the specified NFRs. Some times NFRs are too generic, and some parts of the software could be conditioned by other factors that the user did not specify. In this situation it is more comfortable for the user to allow a manual intervention in the designation of treatments.

8.3.3. Model Editor Tool

Finally, Model Editor Tool (MET) will allow modifying the resultant design model because some kinds of transformations are very complicated to specify and because some requirements will lead to a non-tractable problem. E.g. the user wants that the buttons of the user interface show an alternating blinking green and yellow light. To allow transformations that support requirements like these the framework will require an uncountable number of transformations and transformations that will be too complex. For this reason, the framework allows the user to modify the result and finish the more specific features manually.

8.4. Problems of RDT

Several problems where detected during the design of this framework.

8.4.1. Scalability problems

One of the main concerns about this framework was about the “explosion” of responsibilities that can be detected on a real software project and how they can be managed. Taking into account that we presuppose some human interaction, treating with a huge number of responsibilities could be a problem.

8.4.2. The community acceptance

Difficulties for publishing papers of this framework have been found because of responsibilities. Sometimes the term was misunderstood or confused with the term constraint. Also we found some reticence to accept a change in the way that the MDD is working nowadays.

8.4.3. Implementation problems

It will be reasonable to build a proof of concept of RDT, as was done with the previous work, but implementing the full framework would take so much time and effort.

8.4.4. Ongoing solution

The development of a new framework has begun. It will eliminate the concept of responsibility and, in consequence, the intermediate responsibility model. Also, the new framework will be compliant with the common MDD process.

Implementing the full framework continues being a problem, but when the framework is fully specified I will concentrate my efforts on resolving the problems that concerns to a particular part of it.

All the work done in the design of RDT is being very useful for the new framework.

Part III

Survey:

Usage of architectures and technologies in software development in IT companies and organizations

9. Survey

During the last months I have been building a survey on architectural and technological aspects of software development. The survey has three main sections:

- The habitual practices on software development and the importance given to NFRs
- The desirable or acceptable interaction of the CASE tools
- The Model-Driven Software Development (MDSD) method and the differences between the habitual software development

A transcription of the English version of the survey is available in Appendix A. English and Spanish on-line editions are in the URL:

http://www.lsi.upc.edu/~dsdm-survey

9.1. Motivation

A full background on the academic world was obtained by doing the systematic review (Part I of the Master Thesis). To have a full overview on MDE it is necessary to consider the perspective of companies and organizations. The way chosen to obtain this knowledge was the Survey.

It is expected that the responses of the survey will reinforce the hypothesis of this Master Thesis: “a better support for NFRs is needed in MDE”. Also the responses will guide the development of the new framework in the sense of doing examples and a proof of concept that focuses on the commonly used architectural styles and technologies.

The questions about the interaction in CASE tools will also help in the development of the new framework, to set the boundaries of the interaction with the user.

9.2. Construction method

To construct the survey I have used a specific software that facilitates the creation of on-line surveys, Lime Survey [121]. With this software you can define the survey structure with several types of questions and customize the appearance of the survey.

One of the features of Lime Survey is the ability to set conditions between questions. This ability is good for on-line surveys because each user will see only the questions he/she has to answer (In the printed version of Appendix A, the conditions are in brackets). This idea was exploited in the survey to obtain three surveys in one:

- Survey for people that dos not know MDSD
- Survey for people that know what is MDSD but do not use it
- Survey for people that know and use MDSD at work

The survey was refined several times using testing mechanisms. It was sent to almost ten known people to obtain suggestions. Many of the suggestions were contemplated for the final published version.

9.3. Current state

At the time of writing there are only a few responses to the survey. There is not enough information to do a statistic evaluation. Before next year it is planned to publish a paper with the results of the survey.

Appendix A: Usage of architectures and technologies in software development in IT companies and organizations

With this questionnaire we analyze the usage of architectures and technologies in software development in IT companies and organizations. We are particularly interested to learn about the **impact of non-functional requirements** in your work as an architect, your opinion about the **possibilities and limits of the automation level** in the software development process, and your **knowledge and experiences** about Model-Driven Software Development.

We plan to make public the results of the survey in this website as soon as we have a significant amount of answers. If you want to be notified, please, indicate your e-mail address.

This questionnaire is **anonymous** and it will take you about **15 to 20 minutes.**

There are 50 questions in this survey

Note: *architectural style, technological style, functional requirement, non-functional requirement,* and *model-driven software development* are defined in the glossary.

a. Personal data

1 Name (optional)

Please write your answer here:

2 Company or organization (optional)

Please write your answer here:

3 E-Mail (if you wish to receive the results)

Please write your answer here:

4 Current position in the company or organization *

Please write your answer here:

5 Education to date related to software development *

Please write your answer here:

b. Generic development of software projects

Note: Answer this group of questions **without taking into account** whether the projects were made using MDSD or not.

6 Choose the architectural styles used in your projects: *

Please choose **all** that apply:

- ☐ Service-Oriented Architecture (SOA)
- ☐ 3-layered Architecture
- ☐ Client-Server Architecture
- ☐ Peer-to-peer Architecture
- ☐ Database-centric Architecture
- ☐ Event-Driven Architecture
- ☐ Component-based Architectures (plug-ins, add-ons, extensions, components)
- ☐ Pipe and filter Architecture
- ☐ Mainframe Architecture
- ☐ Model, View, Controller (MVC)
- Other:

7 Choose the type of software developed in your projects: *

Please choose **all** that apply:

- ☐ Web services
- ☐ Web applications
- ☐ Distributed applications based on components
- ☐ Desktop applications
- ☐ Software for mobile devices
- ☐ Software for embedded systems
- ☐ Host applications
- Other:

8 Which of the following technological styles are used in your projects? *

Please choose **all** that apply:

- ☐ Technological style based on Stack solution (e.g. LAMP)
- ☐ Technological style based on Java technologies
- ☐ Technological style based on .Net technologies
- Other:

9 Choose the Stack solution used in your projects: Note: If you checked the option "Other", please specify the operating system, the web server, the data base management system and the programming language used. *

[Only answer this question if you answered 'Technological style based on Stack solution (e.g. LAMP)' to question '8']

Please choose **all** that apply:

- ☐ LAMP (Linux, Apache, MySQL, PHP/Perl/Python)
- ☐ WAMP (Windows, Apache, MySQL, PHP/Perl/Python)
- ☐ WIMP (Windows, IIS, MySQL, PHP/Perl/Python)
- ☐ WISA (Windows, IIS, SQL Server, ASP)
- ☐ OpenACS (Linux/Windows, AOLServer, PostgreSQL/Oracle, Tcl)
- Other:

10 Choose the Java technologies used in your projects: *

[Only answer this question if you answered 'Technological style based on Java technologies ' to question '8']

Please choose **all** that apply:

- ☐ Struts
- ☐ Spring
- ☐ JPA/Hibernate
- ☐ SEAM
- ☐ EJB 2
- ☐ EJB 3
- ☐ JAX-WS
- ☐ JAX-RPC
- ☐ Java Server Faces (JSF)
- ☐ Java Server Pages (JSP)
- ☐ Java Servlets
- Other:

11 Choose the .Net technologies used in your projects: *

[Only answer this question if you answered 'Technological style based on .Net technologies' to question '8']

Please choose **all** that apply:

- ☐ ADO.Net
- ☐ ASP.Net
- ☐ WCF - Windows Communication Foundation
- ☐ WF - Windows Workflow Foundation
- ☐ WPF - Windows Presentation Foundation
- ☐ Spring.Net
- ☐ NHibernate
- ☐ Windows Forms
- Other:

12 Choose the type of data base used in your projects: *

Please choose **all** that apply:

- ☐ Relational
- ☐ Multidimensional
- ☐ Object-Relational
- ☐ Object-Oriented
- ☐ Documental
- ☐ Deductive
- ☐ XML
- Other:

13 Choose the Data Base Management System (DBMS) used in your projects: *

Please choose **all** that apply:

- ☐ MySQL
- ☐ PostgreSQL
- ☐ Oracle
- ☐ SQL-Server
- ☐ DB2
- Other:

Master Thesis: Survey

14 Choose the relevance of the following DBMS capabilities in your projects:

Please choose the appropriate response for each item:

	None	Marginal	Medium	Important	Critical
Stored procedures	○	○	○	○	○
Triggers	○	○	○	○	○
Schema validation (e.g. checks)	○	○	○	○	○

15 Which of the following statements better describes the importance of non-functional requirements to you? *

Please choose **only one** of the following:

- ○ I don't consider them, I focus on the functional part
- ○ I consider them but I don't use them to take important decisions
- ○ They have the same importance as functional requirements

16 Do you use the non-functional requirements to choose between different architectural styles and/or technological styles? *

[Only answer this question if you answered 'They have the same importance as functional requirements' to question '15']

Please choose **only one** of the following:

- ○ Yes
- ○ No

Master Thesis: Survey

17 Choose the relevance of the following types of non-functional requirements on the development of your software projects:

[Only answer this question if you answered 'They have the same importance as functional requirements' or 'I consider them but I don't use them to take important decisions' to question '15']

Please choose the appropriate response for each item:

	None	Marginal	Medium	Important	Critical
Maintainability	○	○	○	○	○
Reusability	○	○	○	○	○
Efficiency	○	○	○	○	○
Reliability	○	○	○	○	○
Usability	○	○	○	○	○
Portability	○	○	○	○	○
Cost	○	○	○	○	○
Standards compliance	○	○	○	○	○
Organizational	○	○	○	○	○

Organizational requirements refer to aspects of the organization where the software system will be deployed.

18 Do you consider other types of non-functional requirements during the development of your software projects?

[Only answer this question if you answered 'I consider them but I don't use them to take important decisions' or 'They have the same importance as functional requirements' to question '15']

Please write your answer here:

19 Do the development tools that you use in your software projects allow you to analyze the compliance with the specified non-functional requirements in different technological styles? *

[Only answer this question if you answered 'They have the same importance as functional requirements' or 'I consider them but I don't use them to take important decisions' to question '15']

Please choose **only one** of the following:

- ○ Yes
- ○ No

20 Which tools do you use to analyze the compliance with the specified non-functional requirements in different technological styles?

[Only answer this question if you answered 'I consider them but I don't use them to take important decisions' or 'They have the same importance as functional requirements' to question '15' *and* if you answered 'Yes' to question '19']

Please write your answer here:

21 Would you like to have tools and/or automatic processes that take into account non-functional requirements? *

[Only answer this question if you answered 'I consider them but I don't use them to take important decisions' or 'They have the same importance as functional requirements' to question '15' *and* if you answered 'No' to question '19']

Please choose **only one** of the following:

- ○ Yes
- ○ No

c. Interaction level

22 For the following tasks of the implementation phase choose the interaction level that you consider more adequate assuming that a hypothetic support tool is available. (1 to 5 as shown) *

1: I wouldn't use any supporting tool to perform this task
2: The hypothetic support tool should ask me before taking any decision
3: The hypothetic support tool should ask me only before taking the relevant Decisions
4: The hypothetic support tool should take the decisions for me but later I would check them
5: The hypothetic support tool would take the decisions for me without further confirmation

Please choose the appropriate response for each item:

	1	2	3	4	5
Generation of the skeleton code	○	○	○	○	○
Generation of the code for a specific technology	○	○	○	○	○

23 For the following tasks of the design phase indicate the interaction level that you consider more adequate assuming that a hypothetic support tool is available. (1 to 5 as shown in the previous question) *

[Only answer this question if you answered 'I consider them but I don't use them to take important decisions' or 'They have the same importance as functional requirements' to question '15']

Please choose the appropriate response for each item:

	1	2	3	4	5
Selection of the architectural style that better conforms to the non-functional requirements of the software system	○	○	○	○	○
Selection of the technological style that better conforms to the non-functional requirements of the software system	○	○	○	○	○

d. Model-Driven Software Development (MDSD)

24 According to your knowledge and skills, which of the following categories do you belong to? *

Please choose **only one** of the following:

- ○ I don't know what is Model-Driven Software Development
- ○ I know the concept of Model-Driven Software Development but I don't use it in my work
- ○ I have used the Model-Driven Software Development paradigm in my work

25 Choose the initiatives you know: *

[Only answer this question if you answered 'I know the concept of Model-Driven Software Development but I don't use it in my work' to question '24']

Please choose **all** that apply:

- ☐ Model-Driven Architecture (MDA)
- ☐ Model-Driven Development (MDD/MDSD)
- ☐ Model-Driven Engineering (MDE)
- Other:

26 Choose the Model-Driven Development platforms you know: *

[Only answer this question if you answered 'I know the concept of Model-Driven Software Development but I don't use it in my work' to question '24']

Please choose **all** that apply:

- ☐ Eclipse EMP
- ☐ AndroMDA
- ☐ openArchitectureWare
- ☐ I don't know any
- Other:

27 Which Model Driven Software Development CASE tools do you know? (Model editors, etc.)

[Only answer this question if you answered 'I know the concept of Model-Driven Software Development but I don't use it in my work' to question '24']

Please write your answer here:

28 In how many projects have you applied Model Driven Software Development? *

[Only answer this question if you answered 'I have used the Model-Driven Software Development paradigm in my work' to question '24']

Please write your answer here:

29 Are the architectural styles used in your MDSD projects different than the ones used in your other projects? *

[Only answer this question if you answered 'I have used the Model-Driven Software Development paradigm in my work' to question '24']

Please choose **only one** of the following:

- ○ Yes
- ○ No

30 Choose the architectural styles used in your MDSD projects: *

[Only answer this question if you answered 'I have used the Model-Driven Software Development paradigm in my work' to question '24' *and* if you answered 'Yes' to question '29']

Please choose **all** that apply:

- ☐ Service-Oriented Architecture (SOA)
- ☐ 3-layered Architecture
- ☐ Client-Server Architecture
- ☐ Peer-to-peer Architecture
- ☐ Database-centric Architecture
- ☐ Event-Driven Architecture
- ☐ Component-based Architectures (plug-ins, add-ons, extensions, components)
- ☐ Pipe and filter Architecture
- ☐ Mainframe Architecture
- ☐ Model, View, Controller (MVC)
- Other:

31 The type of software that you developed using MDSD is different than the type of software developed in your other projects? *

[Only answer this question if you answered 'I have used the Model-Driven Software Development paradigm in my work' to question '24']

Please choose **only one** of the following:

- ○ Yes
- ○ No

32 Choose the type of software developed in your MDSD projects: *

[Only answer this question if you answered 'I have used the Model-Driven Software Development paradigm in my work' to question '24' *and* if you answered 'Yes' to question '31']

Please choose **all** that apply:

- ☐ Web services
- ☐ Web applications
- ☐ Distributed applications based on components
- ☐ Desktop applications
- ☐ Software for mobile devices
- ☐ Software for embedded systems
- ☐ Host applications
- Other:

33 Which of the following technological styles are used in your MDSD projects? *

[Only answer this question if you answered 'I have used the Model-Driven Software Development paradigm in my work' to question '24' *and* if you answered 'Yes' to question '31']

Please choose **all** that apply:

- ☐ Technological style based on Stack solutions (e.g. LAMP)
- ☐ Technological style based on Java technologies
- ☐ Technological style based on .Net technologies
- Other:

34 Choose the Stack solution used in your projects: Note: If you checked the option "Other", please specify the operating system, the web server, the data base management system and the programming language used. *

[Only answer this question if you answered 'I have used the Model-Driven Software Development paradigm in my work' to question '24' *and* if you answered 'Yes' to question '31' *and* if you answered 'Technological style based on Stack solutions (e.g. LAMP)' to question '33']

Please choose **all** that apply:

- ☐ LAMP (Linux, Apache, MySQL, PHP/Perl/Python)
- ☐ WAMP (Windows, Apache, MySQL, PHP/Perl/Python)
- ☐ WIMP (Windows, IIS, MySQL, PHP/Perl/Python)
- ☐ WISA (Windows, IIS, SQL Server, ASP)
- ☐ OpenACS (Linux/Windows, AOLServer, PostgreSQL/Oracle, Tcl)
- Other:

35 Choose the Java technologies used in your MDSD projects: *

[Only answer this question if you answered 'I have used the Model-Driven Software Development paradigm in my work' to question '24' *and* if you answered 'Yes' to question '31' *and* if you answered 'Technological style based on Java technologies ' to question '33']

Please choose **all** that apply:

- ☐ Struts
- ☐ Spring
- ☐ JPA/Hibernate
- ☐ SEAM
- ☐ EJB 2
- ☐ EJB 3
- ☐ JAX-WS
- ☐ JAX-RPC
- ☐ Java Server Faces (JSF)
- ☐ Java Server Pages (JSP)
- ☐ Java Servlets
- Other:

36 Choose the .Net technologies used in your MDSD projects: *

[Only answer this question if you answered 'I have used the Model-Driven Software Development paradigm in my work' to question '24' *and* if you answered 'Yes' to question '31' *and* if you answered 'Technological style based on .Net technologies' to question '33']

Please choose **all** that apply:

- ☐ ADO.Net
- ☐ ASP.Net
- ☐ WCF - Windows Communication Foundation
- ☐ WF - Windows Workflow Foundation
- ☐ WPF - Windows Presentation Foundation
- ☐ Spring.Net
- ☐ NHibernate
- ☐ Windows Forms
- Other:

37 Do you use a particular type of DBMS in your MDSD projects? *

[Only answer this question if you answered 'I have used the Model-Driven Software Development paradigm in my work' to question '24']

Please choose **only one** of the following:

- ○ Yes
- ○ No

Master Thesis: Survey

38 Choose the type of data base used in your MDSD projects: *

[Only answer this question if you answered 'I have used the Model-Driven Software Development paradigm in my work' to question '24' *and* if you answered 'Yes' to question '37']

Please choose **all** that apply:

- ☐ Relational
- ☐ Multidimensional
- ☐ Object-Relational
- ☐ Object-Oriented
- ☐ Documental
- ☐ Deductive
- ☐ XML
- Other:

39 Choose the Data Base Management System (DBMS) used in your MDSD projects: *

[Only answer this question if you answered 'I have used the Model-Driven Software Development paradigm in my work' to question '24' *and* if you answered 'Yes' to question '37']

Please choose **all** that apply:

- ☐ MySQL
- ☐ PostgreSQL
- ☐ Oracle
- ☐ SQL-Server
- ☐ DB2
- Other:

40 Choose the relevance of the following DBMS capabilities in your MDSD projects:

[Only answer this question if you answered 'I have used the Model-Driven Software Development paradigm in my work' to question '24' *and* if you answered 'Yes' to question '37']

Please choose the appropriate response for each item:

	None	Marginal	Medium	Important	Critical
Stored procedures	○	○	○	○	○
Triggers	○	○	○	○	○
Schema validation (e.g. checks)	○	○	○	○	○

41 Choose the initiatives used on your MDSD projects: *

[Only answer this question if you answered 'I have used the Model-Driven Software Development paradigm in my work' to question '24']

Please choose **all** that apply:

- ☐ Model-Driven Architecture (MDA)
- ☐ Model-Driven Development (MDD/MDSD)
- ☐ Model-Driven Engineering (MDE)
- ☐ I don't know
- Other:

42 Choose the platforms that you use on your MDSD projects: *

[Only answer this question if you answered 'I have used the Model-Driven Software Development paradigm in my work' to question '24']

Please choose **all** that apply:

- ☐ Eclipse EMP
- ☐ AndroMDA
- ☐ openArchitectureWare
- Other:

43 Which Model-Driven Software Development CASE tools you use on your MDSD projects? (Model editors, etc.)

[Only answer this question if you answered 'I have used the Model-Driven Software Development paradigm in my work' to question '24']

Please write your answer here:

44 Give us your opinion about the following sentences: *

[Only answer this question if you answered 'I know the concept of Model-Driven Software Development but I don't use it in my work' or 'I have used the Model-Driven Software Development paradigm in my work' to question '24']

Please choose the appropriate response for each item:

	Much worse	Worse	Equal	Better	Much better
The quality of the software architecture obtained by a MDSD process in comparison with the quality obtained using traditional methods is...	○	○	○	○	○
The productivity of using a MDSD process in comparison with traditional methods is...	○	○	○	○	○

45 Which characteristics or functionalities do you think that are currently missing on the platforms and tools of Model-Driven Software Development?

[Only answer this question if you answered 'I know the concept of Model-Driven Software Development but I don't use it in my work' or 'I have used the Model-Driven Software Development paradigm in my work' to question '24']

Please write your answer here:

46 Why haven't you applied Model-Driven Software Development in your projects? *

[Only answer this question if you answered 'I know the concept of Model-Driven Software Development but I don't use it in my work' to question '24']

Please choose **all** that apply:

- ☐ I don't believe in models for software development
- ☐ I don't trust in MDSD
- ☐ MDSD is not mature enough
- ☐ MDSD does not fit to the kind of projects I develop
- ☐ Company policy
- Other:

47 Why MDSD doesn't fit to the kind of projects that you develop?

[Only answer this question if you answered 'I know the concept of Model-Driven Software Development but I don't use it in my work' to question '24' *and* if you answered 'MDSD does not fit to the kind of projects I develop' to question '46']

Please write your answer here:

48 Why do you apply Model-Driven Software Development on your projects? *

[Only answer this question if you answered 'I have used the Model-Driven Software Development paradigm in my work' to question '24']

Please choose **all** that apply:

- ☐ Company policy
- ☐ At a given moment I started to apply MDSD to all my projects
- ☐ I apply MDSD only to some particular kinds of projects
- ☐ I used MDSD in the past but I finally gave up
- ☐ I'm still experimenting
- Other:

49 In which kinds of projects have you applied MDSD?

[Only answer this question if you answered 'I apply MDSD only to some particular kinds of projects' to question '48' *and* if you answered 'I have used the Model-Driven Software Development paradigm in my work' to question '24']

Please write your answer here:

50 Why did you give up MDSD?

[Only answer this question if you answered 'I used MDSD in the past but I finally gave up' to question '48' *and* if you answered 'I have used the Model-Driven Software Development paradigm in my work' to question '24']

Please write your answer here:

Appendix B: Glossary

Architectural style: We understand architectural style as the collection of the main elements that compose the software system and the strategy of communication used between them. Examples of software architectures are: 3-layered architecture, service oriented architecture, client-server, etc. A software system can be designed as a composition of many architectural styles depending on its needs.

ATLAS Transformation Language (ATL): ATL is a model transformation language and toolkit. In the field of Model-Driven Engineering (MDE), ATL provides ways to produce a set of target models from a set of source models.

Complete MOF (CMOF): The CMOF is the meta-metamodel used to specify other metamodels such as the UML metamodel. It is built from EMOF and the core constructs of UML.

Computation Independent Model (CIM): A computation independent model is a view of a system from the computation independent viewpoint. A CIM is sometimes called a domain or business model. The requirements for the system are modeled in CIM.

Domain-Specific Language (DSL): Domain-Specific Languages are languages with very specific goals in design and implementation. A DSL can be either a visual diagramming languages or textual languages.

Ecore: Ecore is a metamodel for describing models and runtime support for the models including change notification, persistence support with default XMI serialization, and an efficient reflective API for manipulating EMF objects generically. It is an implementation of the EMOF standard proposed by OMG.

Essential MOF (EMOF): Is the subset of MOF that closely corresponds to the facilities found in object-oriented programming languages and XML.

Functional requirement: Functional requirements establish the observable behavior that must exhibit the system (calculations, manipulations, listings, evolution aspects, etc.), as well as the data types specification.

Meta Object Facility (MOF): Standard from OMG for defining a platform-independent metadata framework. Some implementations of this standard are: JMI, and EMF.

Metamodel: Description or definition of a well-defined language in the form of a model.

Metamodeling Architecture Layer 0 (M0): The layer that contains the data of the application (for example, the instances populating an object-oriented system at runtime, or rows in relational database tables).

Metamodeling Architecture Layer 1 (M1): The layer that contains the metadata of the application (for example, the classes of an object-oriented system, or the table definitions of a relational database).

Metamodeling Architecture Layer 2 (M2): The layer that contains the meta-metadata that describes the properties that metadata may exhibit (for example, UML elements such as Class, Attribute, and Operation).

Metamodeling Architecture Layer 3 (M3): The layer that contains meta-meta-metadata that describes the properties that meta-metadata can

exhibit. If there were a M4 layer it would look just the same as M3, because M3 is self-describing.

Model: A description of (part of) a system written in a well-defined language.

Model-Driven Architecture (MDA): An approach to IT system specification that separates the specification of functionality from the specification of the implementation of that functionality on a specific technology platform. This approach is adhered to the OMG standards.

Model-Driven Development (MDD): Model-driven development is simply the notion that we can construct a model of a system that we can then transform into the real thing.

Model-Driven Engineering (MDE): Model-driven engineering technologies offers an approach to address the inability of third-generation languages to alleviate the complexity of platforms, and express domain concepts effectively.

Model-Driven Software Development (MDSD): MDSD is based on the construction of a system model that can be transformed, in a systematic and semiautomatic way, into an implementation deployed on one or more software platform technologies. The system model can be unique or can be a combination of models (e.g., UML models). The concept of MDSD is also known with other names that are basically similar: MDA: Model-Driven Architecture; MDD: Model-Driven Development; MDE: Model-Driven Engineering, etc.

Non-functional requirement: Non-functional requirements establish the criteria or global qualities of the software system and set restrictions (internal and external) on the software and the development process. Common types of non-functional requirements are: usability, efficiency and portability.

Object Constraint Language (OCL): OCL is a declarative language for describing rules that apply to Unified Modeling Language (UML) models. It is part of the OMG standards. OCL may be used with any MOF-based meta-model, including UML. OCL is a key component of the OMG standard recommendation for transforming models, the QVT.

Platform Independent Model (PIM): A model that contains no details that have meaning only within a specific platform.

Platform Specific Model (PSM): A model that contains details that have meaning only within a specific platform.

Profile: A profile allows the customization and extension of the modeling language syntax and semantics to define specialized modeling languages for particular domains.

Query/View/Transformation (QVT): QVT is the standard recommendation of OMG to define transformation languages. It is dependent of MOF and OCL standards and it is composed of three sublanguages: QVT-Operational, QVT-Relational, and QVT-Core.

Responsibility: A responsibility contains one or more of the purposes or obligations of one element. Responsibilities are derived from the specification of the system.

Technological style: A technological style is a set of technologies to construct the elements that compose the software system. A technological

style must consider all necessary technological roles of the implementation: platform, programming languages, libraries, technological standards and external services (e.g. database management systems or authentication services). The technologies that take part in a technological style must be able to work jointly.

Unified Modeling Language (UML): UML is a language to specify, visualize, and document models of software systems, including their structure and design. UML can be used for business modeling and modeling of other non-software systems.

XML Metadata Interchange (XMI): XMI is an OMG standard for exchanging metadata information via XML. It can be used for any metadata whose metamodel is expressed in MOF. The most common use of XMI is as an interchange format for UML models.

Appendix C: References

[1] Grup de recerca en Enginyeria del Software per als Sistemes d'Informació, http://www.lsi.upc.edu/~gessi/ *(Last access: June, 2009).*

[2] Llenguatges i Sistemes Informàtics, http://www.lsi.upc.edu/ *(Last access: June, 2009).*

[3] Universitat Politècnica de Catalunya, http://www.upc.edu/ *(Last access: June, 2009).*

[4] D. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," *Computer IEEE*, vol. 39, pp. 25-31, 2006.

[5] I. Sommerville, "Integrated Requirements Engineering: A Tutorial," *IEEE Software*, vol. 22, pp. 16-23, 2005.

[6] B. Kitchenham and S. L. Pfleeger, "Software Quality: The Elusive Target," *IEEE Software*, vol. 13, pp. 12-21, 1996.

[7] D. Ameller, "Assignació de responsabilitats usant AndroMDA," in *Llenguatges i Sistemes Informàtics (LSI)*, Diploma in informatics. Barcelona, Spain: Universitat Politècnica de Catalunya, 2007.

[8] D. Ameller and X. Franch, "Assigning Treatments to Responsibilities in Software Architectures," in *EUROMICRO Conf. on Software Engineering and Advanced Applications (SEAA)*. Lübeck, Germany, 2007.

[9] D. Ameller and X. Franch, "Asignación de Tratamientos a Responsabilidades en el contexto del Diseño Arquitectónico Dirigido por Modelos," in *Workshop on Desarrollo de Software Dirigido por Modelos (DSDM), Jornadas de Ingeniería del Software y Bases de Datos (JISBD)*. Zaragoza, Spain, 2007.

[10] D. Ameller and X. Franch, "Combining Model-Driven Development and Architectural Design in the AR3L Framework," Report in Llenguatges i Sistemes Informàtics (LSI), Barcelona, Spain 2008.

[11] Assignment of Responsibilities in a 3-Layered architecture, http://www.lsi.upc.edu/~gessi/AR3L/ *(Last access: June, 2009).*

[12] M. Glinz, "On Non-Functional Requirements," in *IEEE Int. Requirements Engineering Conf. (RE)*. New Delhi, India, 2007.

[13] ISO9126-1, "Software engineering - Product quality - Part 1: Quality model," in *ISO/IEC*, 2001.

[14] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and Using Nonfunctional Requirements: A Process-Oriented Approach," vol. 18, pp. 483-497, 1992.

[15] B. Kitchenham, "Procedures for Performing Systematic Reviews," 2004.

[16] S. Mellor, K. Scott, A. Uhl, and D. Weise, *MDA Distilled, Principles of Model Driven Architecture*: Addison-Wesley Professional, 2004.

[17] O. Pastor and J. C. Molina, *Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling*: Springer, 2007.

[18] S. Mellor and M. Balcer, *Executable UML: A Foundation for Model-Driven Architecture*: Addison Wesley, 2002.

[19] D. S. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing*: Wiley Publishing, 2003.

[20] A. Kleppe, J. Warmer, and W. Bast, *MDA Explained, The Model Driven Architecture: Practice and Promise*: Addison-Wesley, 2003.

[21] J. Rech and C. Bunse, *Model-Driven Software Development: Integrating Quality Assurance*, 2008.

[22] DSDM: Desarrollo de Software Dirigido por Modelos. MDA y Aplicaciones, http://www.lcc.uma.es/~av/MDD-MDA/ *(Last access: June, 2009).*

[23] MOdeling LAnguages, http://modeling-languages.com/ *(Last access: June, 2009).*

[24] Model Driven Inside, http://www.modeldriveninside.com/ *(Last access: June, 2009).*

[25] Model Transformation, http://www.model-transformation.org/ *(Last access: June, 2009).*

[26] DSM Forum, http://www.dsmforum.org/ *(Last access: June, 2009).*
[27] Metamodel.com, http://www.metamodel.com/ *(Last access: June, 2009).*
[28] The history of conceptual modeling, http://cs-exhibitions.uni-klu.ac.at/index.php?id=185 *(Last access: June, 2009).*
[29] B. Selic, "The Pragmatics of Model-Driven Development," *IEEE Software*, vol. 20, pp. 19-25, 2003.
[30] P. P. Chen, "The Entity-Relationship Model: Toward a Unified View of Data," *ACM Trans. on Database Systems*, vol. 1, pp. 9-36, 1976.
[31] MDA Guide Version 1.0.1, http://www.omg.org/docs/omg/03-06-01.pdf *(Last access: June, 2009).*
[32] A. Kleppe and J. Warmer, "OCL: The constraint language of the UML," *Journal of Object-Oriented Programming*, vol. 12, pp. 10-13, 1999.
[33] Java Metadata Interface (JMI), http://java.sun.com/products/jmi/ *(Last access: June, 2009).*
[34] J. Bézivin, "On the Unification Power of Models," *Software and Systems Modeling*, vol. 4, pp. 171-188, 2005.
[35] An ORMSC Definition of MDA (OMG 04-08-02), http://www.omg.org/docs/ormsc/04-08-02.pdf *(Last access: June, 2009).*
[36] S. Mellor, A. Clark, and T. Futagami, "Model-Driven Development," *IEEE Software*, vol. 20, pp. 14-18, 2003.
[37] Martin Fowler home page, "Language Workbenches and Model Driven Architecture", http://martinfowler.com/articles/mdaLanguageWorkbench.html *(Last access: June, 2009).*
[38] A. Olivé, "Conceptual Schema-Centric Development: A Grand Challenge for Information Systems Research," in *Conf. on Advanced Information Systems Engineering (CAiSE)*. Porto, Portugal, 2005.
[39] T. Kühne, "What is a Model?," in *Dagstuhl Seminar Proceedings*, 2005.
[40] E. Seidewitz, "What Models Mean," *IEEE Software*, vol. 20, pp. 26-32, 2003.
[41] D. Djurić, D. Gašević, and V. Devedžić, "The Tao of Modeling Spaces," *Journal of Object Technology (JOT)*, vol. 5, pp. 125-147, 2006.
[42] C. Atkinson and T. Kühne, "Model-Driven Development: A Metamodeling Foundation," *IEEE Software*, vol. 20, pp. 36-41, 2003.
[43] MOF 2.0 Query / Views / Transformations, http://www.omg.org/docs/ad/02-04-10.pdf *(Last access: June, 2009).*
[44] Eclipse Modeling Framework Project (EMF), http://www.eclipse.org/modeling/emf/.
[45] B. Humm, U. Schreier, and J. Siedersleben, "Model-Driven Development: Hot Spots in Business Information Systems," in *European Conf. on Model Driven Architecture Fondations and Applications (ECMDA-FA)*. Nuremberg, Germany, 2005.
[46] J.-N. Mazón, J. Pardillo, and J. Trujillo, "A Model-Driven Goal-Oriented Requirement Engineering Approach for Data Warehouses," in *IEEE Int. Requirements Engineering Conf. (RE)*. New Delhi, India, 2007.
[47] ATL (ATLAS Transformation Language), http://www.eclipse.org/m2m/atl.
[48] RubyTL, http://rubytl.rubyforge.org/ *(Last access: June, 2009).*
[49] J. S. Cuadrado and J. G. Molina, "A Plugin-Based Language to Experiment with Model Transformation," in *Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS)*. Genova, Italy, 2006.
[50] Model Transformation Framework, http://www.alphaworks.ibm.com/tech/mtf/ *(Last access: June, 2009).*
[51] VIATRA2 Model Transformation Framework, http://dev.eclipse.org/viewcvs/indextech.cgi/gmt-home/subprojects/VIATRA2/ *(Last access: June, 2009).*
[52] K. Czarnecki and S. Helsen, "Feature-based survey of model transformation approaches," in *IBM Systems Journal*, 2006.
[53] Apache Velocity, http://velocity.apache.org/ *(Last access: June, 2009).*
[54] Model to Text (M2T) project, http://www.eclipse.org/modeling/m2t/ *(Last access: June, 2009).*

[55] openArchitectureWare, *http://www.openarchitectureware.org/ (Last access: June, 2009).*
[56] B. Klatt, "Xpand: A Closer Look at the model2text Transformation Language," in *Electronic paper: www.bar54.de/benjamin.klatt-Xpand.pdf (Last access: June 2009)*, 2006.
[57] MDA: The Vision with the Hole?, *http://www.metamaxim.com/download/documents/MDAv1.pdf (Last access: June, 2009).*
[58] B. Hailpern and P. Tarr, "Model-driven development: The good, the bad, and the ugly," *IBM Systems Journal*, vol. 45, pp. 451-461, 2006.
[59] Eclipse Modeling Project, *http://www.eclipse.org/modeling/ (Last access: June, 2009).*
[60] AndroMDA, *http://www.andromda.org/ (Last access: June, 2009).*
[61] Oslo, *http://msdn.microsoft.com/en-us/library/cc709420.aspx (Last access: June, 2009).*
[62] Rational Software Architect, *ftp://ftp.software.ibm.com/software/rational/web/datasheets/rsa.pdf (Last access: June, 2009).*
[63] USE: A UML-based Specification Environment, *http://www.db.informatik.uni-bremen.de/projects/USE/ (Last access: June, 2009).*
[64] Eclipse Modeling Framework Project (EMF), *http://www.eclipse.org/modeling/emf/ (Last access: June, 2009).*
[65] MOSKitt Project, *http://www.moskitt.org/ (Last access: June, 2009).*
[66] MagicDraw, *http://www.magicdraw.com/ (Last access: June, 2009).*
[67] Gentleware, *http://www.gentleware.com/ (Last access: June, 2009).*
[68] Rational Software, *http://www-01.ibm.com/software/rational/ (Last access: June, 2009).*
[69] F. Losilla, C. Vicente-Chicote, B. Álvarez, A. Iborra, and P. Sánchez, "Wireless Sensor Network Application Development: An Architecture-Centric MDE Approach," in *European Conf. on Software Architecture (ECSA)*. Aranjuez, Madrid, Spain, 2007.
[70] J.-C. Bals, F. Christ, G. Engels, and M. Erwig, "ClassSheets - model-based, objectoriented design of spreadsheet applications," in *Int. Conf. Objects, Models, Components, Patterns (Tools-Europe)*. Zurich, Switzerland, 2007.
[71] C. Vicente-Chicote, B. Moros, and A. Toval, "REMM-Studio: an Integrated Model-Driven Environment for Requirements Specification, Validation and Formatting," in *Int. Conf. Objects, Models, Components, Patterns (Tools-Europe)*. Zurich, Switzerland, 2007.
[72] M. Albert, J. Cabot, C. Gómez, and V. Pelechano, "Automatic generation of basic behavior schemas from UML class diagrams," *Software and Systems Modeling*, vol. not published yet, 2008.
[73] S. Konrad, H. J. Goldsby, and B. H. C. Cheng, "i2MAP: An Incremental and Iterative Modeling and Analysis Process," in *Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS)*. Nashville, TN, USA, 2007.
[74] A. Estévez, J. D. García, J. Padrón, C. López, M. Txopitea, B. Alustiza, and J. L. Roda, "Systems Integration Methodology Based on MDA," in *European Conf. on Model Driven Architecture Fondations and Applications (ECMDA-FA)*. Bilbao, Spain, 2006.
[75] E. A. Sánchez, G. Merin, and J. Muñoz, "Hacia un Marco Práctico de Evaluación de Tecnologías de Transformación de Modelos en la Plataforma Eclipse," in *Workshop on Desarrollo de Software Dirigido por Modelos (DSDM), Jornadas de Ingeniería del Software y Bases de Datos (JISBD)*. Zaragoza, Spain, 2007.
[76] J. E. Pérez-Martínez and A. Sierra-Alonso, "From Analysis Model to Software Architecture: A PIM2PIM Mapping," in *European Conf. on Model Driven Architecture Fondations and Applications (ECMDA-FA)*. Bilbao, Spain, 2006.
[77] M. Albert, J. Muñoz, V. Pelechano, and Ó. Pastor, "Model to Text Transformation in Practice: Generating Code from Rich Associations

Specifications," in *Int. Conf. on Conceptual Modeling (ER)*. Arizona, USA, 2006.
[78] A. Kleppe, "MCC: A Model Transformation Environment," in *European Conf. on Model Driven Architecture Fondations and Applications (ECMDA-FA)*. Bilbao, Spain, 2006.
[79] B. Vanhooff, D. Ayed, S. V. Baelen, W. Joosen, and Y. Berbers, "UniTI: A Unified Transformation Infrastructure," in *Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS)*. Nashville, TN, USA, 2007.
[80] T. Mens, K. Czarnecki, and P. V. Gorp, "A Taxonomy of Model Transformations," in *Dagstuhl Seminar Proceedings*, 2005.
[81] H. Giese and R. Wagner, "From model transformation to incremental bidirectional model synchronization," *Software and Systems Modeling*, vol. 8, pp. 21-43, 2009.
[82] MOFLON, *http://www.moflon.org/ (Last access: June, 2009).*
[83] G. Brunet, M. Chechik, S. Easterbrook, S. Nejati, N. Niu, and M. Sabetzadeh, "A manifesto for model merging," in *Workshop on Global integrated Model Management (GaMMa), Int. Conf. on Software Engineering (ICSE)*. Shanghai, China 2006.
[84] Object Constraint Language Specification, version 2.0, *http://www.omg.org/docs/formal/06-05-01.pdf (Last access: June, 2009).*
[85] P. Kosiuczenko, "Specification of Invariability in OCL," in *Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS)*. Genova, Italy, 2006.
[86] D. Costal, C. Gómez, A. Queralt, R. Raventós, and E. Teniente, "Facilitating the Definition of General Constraints in UML," in *Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS)*. Genova, Italy, 2006.
[87] J. Cabot and C. Gómez, "Deriving Operation Contracts from UML Class Diagrams," in *Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS)*. Nashville, TN, USA, 2007.
[88] J. Cabot, R. Clarisó, and D. Riera, "Verification of UML/OCL Class Diagrams using Constraint Programming," in *Workshop on Model Driven Engineering, Verification, and Validation (MoDeVVA), IEEE Int. Conf. on Software Testing Verification and Validation (ICST)*. Lillehammer, Norway, 2008.
[89] Y. Lin, J. Zhang, and J. Gray, "Model Comparison: A Key Challenge for Transformation Testing and Version Control in Model Driven Software Development," in *Workshop on Best Practices Model-Driven Software Development, Int. Conf. on Generative Programming and Component Engineering (GPCE)*. Vancouver, Canada, 2004.
[90] Y. Lin, J. Zhang, and J. Gray, "A Testing Framework for Model Transformations," in *Model-Driven Software Development*, I. S. Beydeda, M. Book, and V. Gruhn, Eds.: Springer, 2005, pp. 219–236.
[91] H. Giese, S. Glesner, J. Leitner, W. Schäfer, and R. Wagner, "Towards Verified Model Transformations," in *Workshop on Model design and Validation (MoDeVA), Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS)*. Genova, Italy, 2006.
[92] D. S. Kolovos, R. Paige, L. Rose, and F. Polack, "Unit Testing Model Management Operations," in *Workshop on Model Driven Engineering, Verification, and Validation (MoDeVVA), IEEE Int. Conf. on Software Testing Verification and Validation (ICST)*. Lillehammer, Norway, 2008.
[93] Epsilon, *http://www.eclipse.org/gmt/epsilon/ (Last access: June, 2009).*
[94] I. Jacobson, G. Booch, and J. Rumbaugh, *Unified Software Development Process*: Addison Wesley, 1999.
[95] L. Chung, J. Mylopoulos, E. Yu, and B. Nixon, *Non-Functional Requirements in Software Engineering*: Springer, 2000.
[96] F. P. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering," *Computer IEEE*, vol. 20, pp. 10-19, 1987.
[97] UML Profile for MARTE, Beta 2, *http://www.omgmarte.org/Documents/Specifications/08-06-09.pdf (Last access: June, 2009).*

[98] H. Espinoza, H. Dubois, S. Gérard, J. Medina, D. C. Petriu, and M. Woodside, "Annotating UML Models with Non-functional Properties for Quantitative Analysis," in *Satellite Events, Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS)*. Montego Bay, Jamaica, 2005.

[99] J. Cabot and E. Yu, "Improving requirements specifications in model-driven development processes," in *Int. Workshop on Challenges in Model-Driven Software Engineering (ChaMDE), Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS)*. Toulouse, France, 2008.

[100] L. M. Cysneiros and J. C. S. d. P. Leite, "Using UML to Reflect Non-Functional Requirements," in *IBM Centre for Advanced Studies Conference*. Toronto, Canada, 2001.

[101] L. M. Cysneiros and J. C. S. d. P. Leite, "Nonfunctional Requirements: From Elicitation to Conceptual Models," *IEEE Transactions on Software Engineering*, vol. 30, pp. 328-350, 2004.

[102] J. I. Panach, S. España, A. M. Moreno, and O. Pastor, "Dealing with Usability in Model Transformation Technologies," in *Int. Conf. on Conceptual Modeling (ER)*. Barcelona, Spain, 2008.

[103] S. Abrahão, J. A. Carsí, M. Genero, E. Insfran, M. Piattini, and I. Ramos, "Towards Quality-Driven Model Transformations: A Replication Study," in *Workshop on Empirical Studies of Model-Driven Engineering (ESMDE), Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS)*. Toulouse, France, 2008.

[104] S. Röttger and S. Zschaler, "Model-Driven Development for Non-functional Properties: Refinement Through Model Transformation," in *Int. Conf. on the Unified Modeling Language (UML)*. Lisbon, Portugal, 2004.

[105] V. Cortellessa, A. D. Marco, and P. Inverardi, "Non-functional Modeling and Validation in Model-Driven Architecture," in *Working IEEE/IFIP Conf. on Software Architecture (WICSA)*. Mumbai, India, 2007.

[106] V. Cortellessa, A. D. Marco, and P. Inverardi, "Integrating Performance and Reliability Analysis in a Non-Functional MDA Framework," in *Workshop on Fundamental Approaches to Software Engineering (FASE), European Joint Conferences on Theory and Practice of Software (ETAPS)*. Braga, Portugal, 2007.

[107] S. Bernardi, J. Merseguer, and D. C. Petriu, "Adding Dependability Analysis Capabilities to the MARTE Profile," in *Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS)*. Toulouse, France, 2008.

[108] M. D. Cin, "Extending UML towards a useful OO-language for modeling dependability features," in *IEEE Int. Workshop on Object-Oriented Real-Time Dependable Systems (WORDS)*. Guadalajara, Mexico, 2003.

[109] UML Profile for for Schedulability, Performance, and Time Specification, *http://www.omg.org/docs/formal/05-01-02.pdf (Last access: June, 2009).*

[110] B. Selic and J. Rumbaugh, "Using UML for Modeling Complex Real-Time Systems," 1998. http://www.dcc.ttu.ee/Automaatika/LAP/ISP0011/umlrt.pdf

[111] H. Wada, J. Suzuki, and K. Oba, "A Feature Modeling Support for Non-Functional Constraints in Service Oriented Architecture," in *IEEE Int. Conf. on Services Computing (SCC)*. Ghent, Belgium, 2007.

[112] L. Zhu and I. Gorton, "UML Profiles for Design Decisions and Non-Functional Requirements," in *Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent (SHARK-ADI), Int. Conf. on Software Engineering (ICSE)*. Minneapolis, USA, 2007.

[113] UML Profile for modeling Quality of Service and Fault Tolerance, *http://www.omg.org/docs/ptc/05-05-02.pdf (Last access: June, 2009).*

[114] J. Jürjens, "UMLsec: Extending UML for Secure Systems Development," 2002. http://www4.in.tum.de/~juerjens/papers/uml02.ps

[115] E. Yu, "Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering," in *IEEE Int. Requirements Engineering Conf. (RE)*. Washington D.C., USA, 1997.

[116] J. Castro, M. Kolp, and J. Mylopoulos, "Towards requirements-driven information systems engineering: the Tropos project," *Information Systems*, vol. 27, pp. 365-389, 2002.
[117] ISO/IEC 25000, "Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE," in *ISO/IEC*, 2005.
[118] D. Ardagna, C. Ghezzi, and R. Mirandola, "Rethinking the Use of Models in Software Architecture," in *Int. Conf. Series on the Quality of Software Architectures (QoSA)*. Karlsruhe, Germany, 2008.
[119] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process (2nd Edition)*: Prentice Hall PTR, 2001.
[120] F. Buschmann, R. Meunier, H. Rohnert, P. Sornmerlad, and M. Stal, *Pattern-Oriented Software Architecture*, vol. 1: A System of Patterns: Wiley, 1996.
[121] Lime Survey, *http://www.limesurvey.org/ (Last access: June, 2009).*