Domain specific modeling and analysis

Jacob, J.F.

Chapter 1

Introduction

In the year 2002 I accepted an invitation to work for four years in research projects at the Centrum voor Wiskunde en Informatica (CWI) in Amsterdam. During that time I would investigate possibilities to leverage scientific research projects with the latest computer science knowledge and skills, such as I could deliver. This thesis and the publications herein are the result of that work.

Only my more successful contributions resulted in publications in this thesis, so the reader might get a rosier impression of the application of new techniques to research projects than was actually the case in practice. However, looking back, it was very well worth the effort, and important lessons have been learned that hopefully help improve future projects.

My main projects for CWI where the OMEGA [OME] and Archimate [Arc] projects. A brief description of these projects is in section 1.1.1 for OMEGA and section 1.1.2 for Archimate. In both projects we worked with several types of model data. The model data is usually a static representation of a state or states in the problem domain. Apart from the precise meaning of the static data, a key problem in projects is usually how to specify the transition from one set of data into another, and, if possible, how to do this in such a way that it is formal but also understandable for the various project participants. Preferably such specifications should lead to insights that guide the implementation of tools. In a typical innovative project such as Archimate or OMEGA, as funded by national governments or international bodies, not only scientific researchers are involved but also representatives from industry. The latter do acknowledge the importance of formal proofs and descriptions concerning the problem domain, but the usual scientific

presentation of formal results is difficult to comprehend without a thorough background in mathematics and formal methods. This is unfortunate because in this way good results may miss the impact they deserve.

This thesis describes several approaches aimed at bridging the gap between science and industry. A recurring theme is the development of demonstration tools, like web applications, that give an insight into formal methods, and that can serve as an intermediate between pure scientists and others. Due to the nature of such tools[1] and due to the limited time available for their development, it is not always possible to give complete results in this way, but this approach is still important because it makes a full formal result, a project deliverable on paper, more acceptable for the non-scientific partners in a project. Another important theme is communication. It is of paramount importance that the scientists and developers in a project communicate well. A thorough study of the core concepts in a project and an agreement on their names and definition is worth the time invested in it. This is closely related to the design of modeling languages, where a good choice of names and abstractions is essential.

Leveraging domain specific concepts in tools helps to make them more accessible and also helps in shortening the tool development time. In every specialized field there are well-known earlier results that can be re-used without the need for a full proof and corresponding full implementation, like it would be necessary if using a general purpose tool. As a very simplified example consider a tool that helps with automating algebra. This tool would not need to prove everything from the ground up, it can use established axioms like $x + y = y + x$ and it does not have to verify the data types of x and y as long as the tool is not abused. These relaxations makes the implementation much simpler and cheaper to develop. A danger in this approach is that users may *unknowingly* abuse the tool, providing input like `5.0 + "1.2"`, where x is a valid number and y incorrectly is a number in string representation. Such errors are usually easy to spot but they stress the fact that domain specific approaches for tools and modeling, as developed in this thesis, are not primarily intended for delivering full proofs or

[1] Tools like web applications are developed using the latest and popular methods and languages so they are familiar to industry, despite the problem that the latest techniques typically do not have a stable formal basis yet.

fully conclusive results, but they are designed in the first place to help with experimenting and with finding results in a timely and cost-effective manner.

With *domain specific* modeling and analysis, as in the title of this thesis, an approach is intended that leverages as much of the earlier existing work in the problem domain as possible. It does this by re–using parts of the languages and formats, typically XML[XML] vocabularies, that are in common use in the problem domain, in order to be able to model and analyse with a formal basis but in a transparant way and in an affordable way with respect to time and cost contraints, concentrating on the original work. A situation often encountered in projects is that even before an attempt to a formal specification is started, there is already a lot of work done on proof–of–concept systems and tools. Domain specific techniques as developed during my projects capture essential concepts and definitions from this earlier work, and give them a name, an abstraction, that is familiar to the early workers. If a truly formal specification, developed at a later point in time, re–uses these concepts, it is better understood and more readily accepted, even if it does not agree in all aspects with early implementations. To be able to reason about the captured concepts algebraically is desirable, and this is a prime example of the usefulness of the transformation capabilities of the techniques introduced in this thesis. However, the main use of the transformation techniques developed in my projects is to translate from a model with domain specific elements to a model that is suitable for other purposes, like a graphical display for visualization or simulation. This improves the level of understanding and communication considerably.

1.1 Problem statement

In order to introduce the more general problem statements, I will first describe the research projects and the problems encountered there.

1.1.1 The OMEGA project

The OMEGA[2] project was a 3-year IST project, IST-2001-33522 OMEGA, in which the CWI, my employer at the time, participated as a research partner.

[2](`http://www-omega.imag.fr/index.php`)

The full title of the project was Correct Development of Real-Time Embedded Systems. Besides research partners there were also several industrial partners in the project, and it was sponsored by the European Commission. As a result of this profile the project aimed to achieve not only theoretical results but also some results that have direct practical benefits, as shown by the official aim of the project that is stated *Definition of a development methodology in UML for embedded and real-time systems based on formal techniques* on the project website. The research partners were teams from VERIMAG from France, also acting as project coordinator, Christian-Albrechts-Universität from Germany, University of Nijmegen from The Netherlands, OFFIS from Germany, The Weismann Institute from Israel, and Centrum voor Wiskunde en Informatica from The Netherlands. The industrial partners were EADS SPACE Transportation from France, France Telecom R&D from France, Israeli Aircraft Industries from Israel, and The National Aerospace Laboratory from The Netherlands.

Project OMEGA achieved many results, in the form of publications but also in conferences, workshops and standard contributions to UML 2.0. Because of the pluriformity of the OMEGA work, there were several work packages: Modeling, System Verification, Synthesis, Development Methodology, and Applications. I started working in Modeling, but soon I directed most of my efforts at System Verification. There were also a few contributions for the Development Methodology, such as the coordination language UnCL from chapter six.

In the OMEGA project the Unified Markup Language (UML) is used for modeling, and as a basis for verification. However, UML itself does not have a formal semantics, there is no mathematical definition of UML. This is not an omission in UML but one of its strong points because it gives more freedom in designing and using UML models, which would be harder if UML, for instance, insisted rigidly on a certain model of execution. Instead of incorporating a formal semantics, UML semantics is given by various UML tools, as encountered in the project. There are tools for model building and model checking and simulation.

An important OMEGA result is the development of the OMEGA Kernel Model language. It is a subset of the UML language, capturing core UML concepts that are important for the users in OMEGA. The Kernel Model is used as a reference point for discussions and comparisons of the various verification tools in the project. It incorporates UML extensions for real–time software.

The consistency problem

The OFFIS and VERIMAG teams worked on model–checking tools and the university partners worked with PVS, a theorem prover. These different tools all have different internal formats and unfortunately this resulted in a *consistency problem*, which turned out to be a major challenge. The problem with the tools that we used is that they have internal details that are not part of the model, for instance the use of certain stacks and tables for namespace administration purposes in the software. This leads to practical problems with the consistency of the results acquired with the tools, because the inner workings of the tools differ and it is not feasible to translate semantics from one tool to the other, and to relate these back to the original model, in a consistent way. The Kernel Model semantics is being mangled by adhering to a specific internal tool format and this has damaging effects on the consistency. It is very hard to explain the semantics of a model when using another, specific, semantics.

The CWI team contributed by defining an *abstract* semantics for the Kernel Model. An *abstract* semantics may function as some sort of bridge between different more concrete semantics. In order to remedy some of the consistency problems the CWI team decided to investigate the possibility of a proof–of–concept tool that provides an implementation for the abstract semantics of the OMEGA Kernel Model. The goal was to achieve a complete separation between the event–based operations and the primitive operations. An example of an event–base operation is a method call in OOP software, an example of a primitive operation is the addition of numbers. We also wanted to separate all operations from the scheduling in the executing environment. Chapter 2 presents this work.

1.1.2 The Archimate project

During my stay at CWI I also put a lot of work in the ArchiMate[3] project, a research initiative that aims to provide concepts and techniques to support architects in the visualization and analysis of integrated architectures. The Archimate consortium consists of ABN AMRO, Stichting Pensioenfonds ABP, the Dutch Tax and Customs Administration, Ordina, Telematica Institute, Centrum voor Wiskunde en Informatica, Radboud University of Nijmegen, and the Leiden Institute for Advanced Computer Science. One of

[3](`http://archimate.telin.nl`)

the results of Archimate is a book publised by Springer with the title *Enterprise Architectures at Work*, and I am one of the authors of that book. The Archimate language developed for enterprise architectures has been adopted as a standard in the Netherlands, Belgium and Luxemburg.

During the project several enterprise architecture description languages were developed, where each language was intended for different stakeholders. A language intended to describe a complete architecture would be too large, and the resulting model would be far too complex. With their own specific language the stakeholders could create a model of an enterprise architecture that would model the parts they were interested in, while abstracting from other parts. Such languages need to capture properties of the system in their bare essence without forcing the architect to include irrelevant detail. The models created in Archimate with these languages were primarily intended for visualisations and simulations, there was no deep investigation into semantics like in the OMEGA project. The work in Archimate was more one of language design than language analysis. An appropriate level of abstraction for the description languages was required, and during the project the languages were subject to change, while searching for such an optimal level in an iterative design process. To complicate matters, since there were different stakeholders in the project with different interests and priorities, their requirements led to very different languages, resulting in a consistency problem similar to that in the OMEGA project. However, in Archimate the consistency problem was of lesser importance, since a unified semantics was not an important goal.

The adaptation problem

The Archimate project developed and used several tools, and the continual rapid changes of the languages posed several problems for the tools that had to work with them. Especially in the early stages there was an *adaptation problem* and this was a bigger problem in Archimate than the consistency problem. The tools had to be able to adapt themselves to new versions of the model languages used, and they had to be able to do that quickly and without too much effort during the course of the project. The model languages used in Archimate were XML[XML] languages, called *vocabularies* in XML terminology, complete with XML *schemas* for the language definitions. If vocabularies are often subject to change, it is best to concentrate on the schemas when developing tools. This is the standard approach when devel-

oping XML tools in such circumstances and it was generally followed by the Archimate tools. The goal is to develop tools that take a schema as input, creating a new tool as it where. This virtual new tool can then handle a model in the schema's language, making the original tool rather independent of the specific language used. This approach is not easy and not straightforward, since development has to take place on the basis of a meta–language rather than a final language that can be used immediately for testing purposes. During development it would be hard to envision what the final tool would be like, adding extra uncertainties to the development process.

As one of the participants in the Archimate project, the CWI team drew attention to other XML work being done at CWI and suggested to investigate if new developments there could be used in Archimate. It was at this point that I joined Archimate to see how I could contribute with XML language design and tool development, primarily focusing on the adaptation problem. Part III of this thesis bundles the work in the Archimate project.

The practical problems encountered in the projects lead to the problem statements of my work:

- How can the consistency of project results acquired with various different tools be improved upon? This *consistency problem* is first introduced in section 1.1.1.
- How to develop tools for a project while the underlying modeling languages are still in flux, being designed and changed in an iterative process? This *adaptation problem* is first introduced in section 1.1.2.
- How can project results be communicated well to other project stakeholders, and how can the design of model languages help in this respect?
- How to create a common language of discourse that is still close to the semantics modelled, again with the design of model languages in mind?
- How can modern techniques in software design and programming be leveraged in research projects? Not as a theoretical research topic, but to enhance the project practically, making use of the latest developments. For instance in the area of web-based systems, protocols, and languages, what is hype and what is useful?

- As a specific example of a hyped technique: Can the definition of new XML vocabularies, defining domain specific XML languages, help in a scientific research context?

- What extra contributions to a typical research project, lasting three or more years, are possible with a domain specific approach? Does it open up new ways of getting results, does it bring new insights?

- How valuable are domain specific techniques? Are they only suitable for simulations and demonstrations or can they also help to obtain more formal results?

- Can domain specific tools be developed and used in the timespan of only a few years as is usually the case in a typical research project?

1.2 Objectives

There are several objectives my work tries to achieve. First of all the practical objectives of immediate use in the projects I was involved in: to solve the consistency problem and the adaptation problems from section 1.1, or at least ameliorate them. This is part of the more general objective to find answers to all the other problem statements from that section.

Another, more long-term, objective is to bridge the gap between formal methods and mathematics on the one hand, and software engineering practice on the other hand. This should lead to a better theoretical basis for UML and other models, and ultimately it should lead to software engineering based on sound formal approaches.

Software engineering today roughly uses three types of models. With increasing formality they are: programming language level models (API's) with written comments, standardized diagrams like UML, and formal specifications. This thesis advocates the use of the latter, but its use is still very rare in industry. Reasons why formal specifications are not popular are that many developers would have to be better trained mathematically, and scaling to real–life size systems has not been accomplished often enough. Also, communicating formal models is complicated since there is a problem of choice. There are many formal methods to choose from, and each has its own notations and techniques.

One objective of the use of domain specific techniques in my work is to turn models that have a feeble formal basis into models that are better suitable for formal methods. Even when this does not immediately lead to a full–fledged formal specification, the results do bring much insight and starting points to arrive at such an enhanced specification later.

But perhaps the most important objective of my work has to do with the human aspect: the domain specific techniques help with understanding results, with analysis, with discussions and with communication. Results can be presented with concepts and definitions that have familiar names for everyone involved. The importance of an excellent mutual understanding and a high level of communication is paramount in research projects.

1.3 Approach

To explain the approach of my work, let me start with a summary of it. The research starting point in this thesis is operational semantics, taken as the foundation to understand systems. This is enhanced with term rewriting techniques in order to describe behavior. In order to facilitate the term rewriting, several pattern matching techniques have been developed that are capable of working with modern data formats like XML[XML]. It turns out that these techniques are very useful for dynamic aspects like simulations and visualisations, where they have been succesfully applied, while the underlying operational semantics, or at least the possibility to envision a clear route to such, provides a good understanding of the whole. In what follows I shall give more background to the research approach, using a lot more lines than this summary, but I wanted to present the summary here first to guide the reader with respect to the direction of the work.

A good approach in research based on other research, is to keep the good things and remove the bad, and to add new good things. This seems obvious and this is the approach chosen in my work. However, to use this approach, one has to identify first what is good and what is bad. This may look trivial, but in computer science, which is a relatively young field, good and bad are not so well–established yet and it is hard to get many experts to agree on a certain topic.

I should note here too that all research described here is conducted in the context of projects. This influences the research approach because this means that there have to be things like feasability studies and the research

always has to keep project goals in mind.

Part of my research approach was to look for promising new techniques and how they could be applied to the project topics. If a technique looked promising enough then I would design and develop a proof–of–concept tool. A presentation to research partners in the project would then provide valuable input from them about the usefulness and suitability. While not a research question or goal in itself, it was very interesting to find out what others, with a different background, had to say about new development techniques and systems. Not every new technique or approach received a warm welcome, even though it was very popular in the world of development specialists. My domain specific approach was also received with healthy scepticism, but it became readily accepted when application of it in the OMEGA and Archimate projects addressed several research questions and fulfilled several research goals.

In the research projects several different kinds of models were used. Many of these models were UML models like class diagrams, message sequence charts, and use cases. Usually, the complexity of a system is such that many different models are needed to model it. This was also the case in the projects, because the projects wanted to achieve practical results and several real–world systems were under investigation. Each different model is used to describe certain aspects of a system, where only parts of the system important for a certain stakeholder are modeled, and other parts of the system are ignored or modeled in much less detail. There is an analogy with blueprints for a building since there we see different ones for the electricity system, the plumbing and the concrete structure. Such modular design and separation of concerns are all very nice indeed, but it is of paramount importance that the different models are consistent. An ideal plumbing system with very desirable properties is useless if the building is not prepared for it. How to arrive at a consistent set of models is the *consistency problem*.

In order to solve the consistency problem, the UML community devotes much research to meta-modeling techniques. The idea is to define a core model and to be able to derive all other models from it and to be able to integrate existing model types. Unfortunately this does not address a major shortcoming of UML: it being unable to provide consistent analysis tools.

Most existing tools as used in the projects, are based on rather traditional techniques and classic ways of dealing with classes and inheritance and other object oriented paradigms (OOP). The tools themselves are written in traditional and well–known programming languages, like C++ and Java. They

are being designed with a rigid top–down design using mostly imperative and OOP techniques. This ties them closely to the model of execution of the programming language chosen and to the intricate details of the compilers used, and these ties are generally incompatible with the chosen core model. Their design and implementation makes the tools rather big and unwieldy to use in novel circumstances, like the introduction of real–time aspects. During my work I kept looking for modern techniques that could be of assistance here. I was also looking for small tools rather than big ones, looking for a combination of small tools that could be better than their sum. Another aspect is the way that tools may exchange models. In order to be able to exchange models an XML vocabulary has been designed by a consortium of UML users, and this XML vocabulary was called XMI. XMI can be seen as a common collection of structures and names and definitions that the various UML tool vendors agreed upon. This leads to the idea of also using it as a basis for analysis techniques and even for formalisations of behavior, since a recurring problem in these is often the establishment of a common language of discourse and good set of definitions that is commonly understood. XMI is very complete but because of this unfortunately also very complex, and less complex solutions were needed for analysis and for dealing with behavior. In the projects I kept looking around for new developments to find such solutions.

With respect to useful specific "latest" computer science techniques, I have used a *dynamic programming language* to be able to provide an executing environment for the various models, and I have chosen *data–centric techniques* to arrive at open and transparant systems with interchangeable data. The choice of a popular modern dynamic programming language proved to work out well, since it was capable of providing more flexible solutions in a shorter time span than would have been possible with traditional languages like C++ and Java. It also provided us with very up–to–date libraries for working with XML and other structured data, where we would have had to wait a significant time period, like months, for similar C++ and Java libraries.

I would like to note here, perhaps again, the importance of taking the existing original structured data, such as XMI, as starting point. This increases the level of trust and understanding in the new, smaller, more formal, model. It also makes validation easier and it can better be verified how the new model relates to the old situation. It is important that familiar names re-appear, familiar structures re-appear, and in the case of an executing model

the same familiar execution steps can be recognized. In an ideal situation one should start with formal specifications, and be free in the choice of names and concepts, but in reality this is not always possible. For instance, Project Management may have decided to use certain UML tools, or to use certain existing software libraries, for reasons that are not always disclosed and anyway beyond the scope of this discussion. Such circumstances however have to be accepted as part of project reality, and I have encountered them in every single project I have been involved in during my twenty–five years in ICT.

While the UML community spends much effort on meta–modeling techniques, my approach concentrates on the integration of models by trying to find similarities while avoiding as much as possible having to put a tree hierarchy on the models. Complementary to the meta–modeling, which is a top–down approach, the domain specific techniques give a bottom–up approach to arrive at an adequate model core. Or, if a single core can not be achieved, the approach still provides methods to relate models to each other, based on an improved mutual understanding of domain specific notions and concepts.

1.4 Working with XML and other structured data

Models, formulas and other data are nowadays often expressed in XML [XML]. It is believed that next–generation programming systems will have computer programs stored as XML or XML–like documents, to increase interoperability, the goal being that data and meta–data can be represented and processed uniformly [Wil05]. XML is seeing an continually increasing use as the format of choice for modeling language, and it is now the most popular choice. A large part of the thesis is about using XML, and about an XML extension called the Rule Markup Language (RML), described in Chapter 3, in particular. It is shown how to define XML languages, with the emphasis on XML for formal methods, and approaches and methodologies are discussed. With RML it is possible to define rule–based transformations of XML in XML itself, and more importantly, this can be defined in the XML *vocabulary* for the topic at hand itself. RML uses the general technique of *pattern–matching* and *variable–binding*, known from the world of regular

expression tools like Perl, where in the case of RML the patterns match XML–parts. These patterns are also expressed with reuse of the domain specific XML vocabulary of choice. Variable bindings with domain specific data can be stored and used at a later time to modify or create other data. An important result in my work is that the freedom given by this approach makes it possible to study and demonstrate formal methods and their applications to models expressed in XML without any restrictions due to the design or implementation of the underlying tools such as modelcheckers and theorem provers.

Besides RML I introduced two other XML techniques to the projects I was involved in: AML [Jaca], see section 8.3, a simpler representation for XML for presentation purposes that is also used to be able to create XML with a simple text-editor, and OOXML, an object-oriented databinding for XML in a high–level scripting language. Like RML, AML and OOXML proved to be very useful to get various work with XML done in a timely fashion in typical research projects.

The pattern–matching and variable–binding approach taken for the XML case with RML can also be applied to other structured data, like text–based notations for formulas. For this purpose ATL has been developed, a wildcard-matching technique for structured text with an as-simple-as-possible design that has a much lower learning curve than typical classical regular expression libraries like those found in Perl, making it applicable without having to learn a full programming language. As a practical example of ATL, a web application is developed that assists with proofs using the tableau method, and a non-trivial proof is derived for the OMEGA project.

1.5 Structure of the thesis

Because of the nature of my work in the projects, the following chapters in this thesis are a number of publications, where every paper forms a chapter by itself. So far the presentation in this thesis has been from abstract to more concrete, but in this section I will revert to a more general bird's eye view of the publications, relating them to each other and to the problem statements, the objectives, and the chosen approach.

There are several scientifically refereed publications, they are:

- Chapter 2. RML and its application to UML. Author: Joost Jacob. Published by Springer in the ISOLA conference proceedings in the

series Lecture Notes in Computer Science, volume 4313, year 2006. [Jac04a]

- Chapter 4. The OMEGA Component Model. Author: Joost Jacob. Published by Springer in the journal Electronic Notes in Theoretical Computer Science, volume 101, year 2004, pages 25-49. [Jac04b]

- Chapter 8. Enterprise Architecture Analysis with XML. Authors: Frank de Boer, Marcello Bonsangue, Joost Jacob, Andries Stam, Leendert van der Torre. Publised by the IEEE Computer Society in the 2005 HICSS conference proceedings. [dBBJ+05]

- Chapter 9. A Logical Viewpoint on Architectures. Authors: Frank de Boer, Marcello Bonsangue, Joost Jacob, Andries Stam, Leendert van der Torre. Publised by the IEEE Computer Society in the 2004 EDOC conference proceedings. [dBBJ+04]

- Chapter 10. Using XML Transformation for Enterprise Architecture. Authors: Frank de Boer, Marcello Bonsangue, Joost Jacob, Andries Stam, Leendert van der Torre. Publised by Springer in the ISOLA conference proceedings in the series Lecture Notes in Computer Science, volume 4313, year 2006. [SJdB+04]

The following chapters are grouped into three parts. Part I introduces RML and its tool support, and contains a paper with results in the OMEGA project. Part II is about work on component models in OMEGA and in distributed environments and introduces another pattern matching technique similar to RML as it was used in OMEGA. Part III is also about models and analysis, but here it is enterprise architectures that are modeled in the Archimate project and RML returns as it is used for their analysis.

Part I is named **RML, a tool for model analysis**. In chapter 2 it starts with a paper titled *A Rule Markup Language and Its Application to UML*[Jac04a]. In this paper RML is introduced and an application to UML models is exhibited. This was my first example where a domain specific technique was succesfully applied. Chapters 2 and 3 contain the main introduction to RML. In the OMEGA work described in chapter 2 we were able to demonstrate that models could indeed be executed based on the abstract semantics we designed. This was important since the abstract semantics of the OMEGA Kernel Model helped to relate the other results in the project

to each other. Also in Part I, in chapter 3, is the RML Tutorial, with examples of all kinds of XML transformations and how to perform these with RML. Part I lays a foundation for the rest of the thesis. RML was used in the majority of my work, and the pattern matching and term–rewriting ideas from RML did play an important role in the rest of it.

Part II has the title **Component Models and Analysis** and consists of four chapters, chapters 4 to 7. Chapter 4 is a paper that reflects the CWI contribution to the OMEGA project with respect to component modeling in UML. Several ideas from the paper can be found in UML standards that appeared later, starting with UML 2.0, for instance the way to model component ports. In Chapter 5 is an OMEGA publication called *Component Coordination in UML*. It has soms overlap with Chapter 4 because it also uses the OMEGA modeling, but it is focusing on *coordination* of components. Chapter 6 is a publication from the Software Engineering department of CWI, SEN report E0511 from 2005, titled *The unified coordination language UnCL*. It is a fusion of my work in OMEGA on components and the work of my colleague Juan Guillen Scholten at CWI on distributed channels, resulting in a coordination language. Chapter 7 is a CWI publication titled *ATL Applied to the Tableau Method*. This paper shows a novel technique that was used in OMEGA to aid in the proof of a software property. The software was modeled with the OMEGA kernel model from chapter 4 but instead of transforming model data in XML, here we wanted to transform formulas with statements about the models. The ATL approach resulted in additional insights, enhancing earlier proofs that were performed in OMEGA using more conventional methods. Part II shows a progression from static models to more dynamic models and their analysis, with a few digressions in order to explain the techniques used.

Part III consists of three papers on enterprise architectures and is titled **Modeling and Analysing Architectures**. Chapters 8 and 9 are papers with the titles *Enterprise Architecture Analysis with XML* [dBBJ+05] and *A Logical Viewpoint on Architectures* [dBBJ+04]. Chapter 10 is the paper titled *Using XML Transformation for Enterprise Architecture* [SJdB+04]. With respect to my contribution to these papers, the results build on the experience gained with models and analysis in part I and part II, but since they are all papers from the Archimate project and their common theme is enterprise architectures, these papers are presented last and bundled together.

My contribution to Archimate consists of XML language design for business processes and their visualizations and simulations and especially the

RML tool for performing transformations on models in XML. Tools for visualization or simulation use RML for the necessary XML transformations. With RML it becomes practically feasible to tune XML languages to the desired goals in an iterative process, while still using the same tool for visualization and the like, without having to recompile or rebuild the tool. The data–centric nature of the RML tools is helpful in this respect: as much logic and behavior as possible is stated in rules and scripts, removing the need to program them in a much more lower level programming language. Language changes are easy to incorporate in the RML rules, since those rules are as close to the language itself as we could design. RML makes it readily possible to transform systems described in one language to another, to analyse and query systems, and RML also provides an executable framework wherein the dynamic behavior of systems defined in the languages can be quickly tested and analysed, before committing too much resources to the development of fully optimized and specialized tools. The RML contribution to the Archimate project is also described by me in chapter 10 of the Springer book *Enterprise Architecture at Work* [ea05].

Since several chapters contain complete papers as published, some chapter contents have a little overlap. This overlap is not removed but preserved in order to support the reader when reading a chapter by itself, without having to direct the reader to other parts of the thesis, for instance for a short introduction of RML.

1.6 Conclusion

The most successful domain specific approaches in the research projects I have contributed to were the development of new XML vocabularies for modeling and analysis purposes, and the RML and AML tools for handling the new XML that was created with the new vocabularies.

Developing new XML vocabularies has been beneficial in both the OMEGA and Archimate projects. The new XML vocabularies formed a basis for tool development and also for discussions of various data–related topics, both static and dynamically. AML made it possible to use the new XML vocabulary in such discussions in a readable form. For instance, discussions about the flow of events in the OMEGA kernel model could be illuminated

with simple classes and objects represented in AML and they could even be dynamically executed with a tool for demonstration purposes. The domain specific techniques did help to achieve a much higher level of communication and understanding between the various project members.

Looking back, especially the development of RML was very helpful in producing results in the OMEGA and Archimate projects. The existing XML tools that were in use in industry at the time were too cumbersome and producing tools with them would take too long in a research project setting. However, today RML has not attained a top–rate status when it comes to XML tools. Reasons are that the CWI research institution where it was developed is not a commercial software house, meaning it has no incentive nor facilities to produce industry–strength competitive software, and it does not have a marketing department that can draw attention to its products. There is also the fact that the main RML virtues are its simplicity and minimalism, and those virtues do not have much marketing value in today's ICT world. Anyway, it is not the tool itself, but its underlying principle of using a domain specific approach, that I consider an important result of my work.

Domain specific languages and models and methods deserve attention from the scientific world. They are popular and they are found everywhere. As an example, consider the HL7 [SRMM00] [7] language that is used in the healthcare domain. The aim is to support hospital workflows through electronic messages exchange between administrative, logistical, financial as well as clinical processes (for instance to send patient data to a radiology department). While it initially used a proprietary (non-XML) syntax, the most recent version uses only XML as a syntax for messages.

Almost all hospitals in The Netherlands use HL7 messages and documents for exchanging medical information. A large number of tools are available for developers, implementers and users of HL7, mostly concentrating on simulation, editing, viewing and validating the XML specific vocabulary of HL7. For these tools, either their formal basis seems feeble, or, as in the case of commercial products, their formal basis is undisclosed. Most of the tools that are available commercially to work with HL7 are complex, often not satisfactory, cumbersome and require users to follow courses to even learn to work with them.

In my CWI research projects I concluded that several small tools may together produce a better result than one large system, on the condition that their results are consistent. But this requires more time spent on design and

discussions, and a less strict product–manager–like attitude. In my opinion, theoretical and hard–core computer scientist are definitely able to give a valuable contribution to the use of various domain specific languages. But they are sometimes not invited when I feel they should be. As a result, several real–world domain specific languages have a basis that is not as formal as would be desirable. And the other way around, computer scientists are sometimes not interested in a domain specific language project, being afraid of being dragged into tool development with little scientific value. This situation is unfortunate for both sides, and I feel there are many improvements possible, for instance the use of domain specific techniques with a design like I used for RML.

Why do projects spend so much time and effort to define domain specific languages instead of first defining a formal specification and then building a language on top of that? With a formal specification in hand, designing a domain specific language is much more robust and also simpler, even when the foqmal specification is only halfway ready. There are several reasons. Unfamiliarity of managers, directors, and other decision makers with formal methods is one. Scarcity of mathematically schooled developers is another. Yet another reason is that there is often an earlier body of work, for instance an existing implementation of part of the desired functionality, and management decides that it is cost–efficient and wise to reuse it. All such reasons obstruct a good design of a domain specific language. This is unfortunate, since it is my experience that especially in the early stages of a project, results are obtained faster when working with a well–designed domain specific language for the data rather than by taking the traditional route of defining the data in a full fledged programming language, for instance an object oriented class library in Java or a complex datastructure in C. And still, this is what happens often when the decision is made to reuse existing software or an existing tool, thereby making a formal basis problematical.

Modifications to a data design are easier when it is more separated from the tool implementation, and such modifications are frequently needed in the early stages of a project. A modification like changing a naming convention in the data may seem unsignificant but it is not, because the data language serves as a language of discourse in project discussions. A new naming convention for a group of data elements is much simpler to implement within a domain specific language than in an object oriented class library, and this is just another example of why it is advantageous to use a domain specific approach.

Most research questions from chapter one, the Introduction, have been anwered in the publications in the later chapters. It has been found possible to introduce new techniques in scientific research projects in a beneficial way. Mainly to produce tools for visualisations and simulations, but also contributing in a more fundamental way and resulting in proof–of–concept tools. On several occasions the tool developement led to fruitful discussions and new ideas. Usage of XML and the design of new XML vocabularies proved to be valuable. The development of new small tools to work with the XML also proved to be worthwhile, working on the XML itself or for instance to translate from XML to PVS. It was sometimes possible to combine a set of small tools resulting in a whole that was better than their sum. This is reminding us of the well–known ways a combination of tools in the UNIX world would be used to produce new tools, an art that has become less popular in these days of big computer languages like Java and C# and their massive development environments. Making use of new dynamic programming languages and data–centric techniques, we were able to develop such new small tools within the timespan of the projects, and here I feel that it was important that there were not too much restrictions on the implementation. It was important that the programmer was free in the choice of a programming language and in the design of the tools. Programmers need freedom to be creative and productive, and it seems that the better the programmer, the more freedom is necessary. On first sight, this principle advises against the use of formal specifications, but I believe this is not the case. If the formal specification is able to stay close to the world of the programmer, using concepts and definitions the programmer is familiar with, then the insights acquired from the mathematics are a joy to work with. The development of domain specific techniques and their application helps to bring formal methods closer to the many existing and popular domain specific languages that are already being used on a large scale but lack a real formal basis.

Finding new techniques, determining their usefulness, and introducing them to projects, remains a considerable task. Some new techniques proved to be helpful, like the XML modeling that could quickly yield new tools, while other new techniques turned out to be mostly hype and they could not withstand scrutiny by scientific minds.