Industrial Experiences from Multi-Paradigmatic Modelling of Signal Processing

Håkan Burden
Computer Science
University of Gothenburg
Gothenburg, Sweden
hakan.burden@gu.se

Rogardt Heldal
Software Engineering
Chalmers University of Technology
Gothenburg, Sweden
heldal@chalmers.se

Martin Lundqvist
Baseband Research
Ericsson AB
Gothenburg, Sweden
martin.lundqvist@ericsson.com

ABSTRACT

Embedded software is often composed of interacting domains. A common problem is that the implementation intertwines the different domain solutions with each other and the platform-specific details. The result is a code mass that is hard to understand, maintain and reuse. We report on an effort to overcome these problems by using a domain-specific executable modelling language for each included domain. The application was delivered for the Ericsson LTE-A uplink test bed as part of the 4G telecommunications system that was presented at the Mobile World Congress in Barcelona, February 2011. The requirements for the delivered software included efficient real-time performance for signal processing on new hardware as well as a firm non-negotiable delivery deadline. Our results show that the chosen modelling languages allowed independent implementation and validation of each domain. Neither did the integration of the separate solutions imply additional problems.

Categories and Subject Descriptors

D.2.13 [**Software Engineering**]: Reusable Software—*Domain engineering*; I.6.5 [**Simulation and Modeling**]: Model Development—*Modeling methodologies*

General Terms

Languages, Performance.

Keywords

Executable Software Models, Digital Signal Processing, Telecommunications Industry, Case Study

1. INTRODUCTION

Embedded software applications in industry are often composed of an interacting set of solutions to problems originating from different domains. Although these problem domains may be naturally separated, and initially specified by different expert designers using different appropriate methods, the actual implementation of the combined application is often delegated to programmers using general program languages. The programmers are forced to add program language-dependent details in their manually produced code, including optimizations for the current hardware. The result is a mix-up of the desired functionality and structure of the system together with the hardware-specific details, all intertwined in the syntax of the program languages used for implementation.

We decided to explore the possibilities for using multiple modelling languages in implementing the channel estimation of the LTE-A uplink test bed of a 4G telecommunication system [8]. The requirements on such an application includes unconditional real-time performance for calculations on synchronous data and the contextual determination of when signals shall be sent and processed as well as operation reliability.

In our contribution we show that it is possible to approach such a system by identifying its different domains - based on their main properties and respective needs for expression - and the interfaces between them. Each domain can then be implemented independently by using an executable and translatable modelling language.

The related work is presented in section 2, followed by our motivation in section 3. In section 4 we give the necessary background about the investigated domains and what we consider suitable modelling languages for each domain. In section 5 the delivered application is described together with the implementation process. The outcome of the process is then found in section 6 which is followed by our discussion, section 7. Finally, we conclude and propose new questions for further research in section 8.

2. RELATED WORK

A substantial part of research has been reported for combining multiple modelling or domain-specific languages from the perspective of meta-modelling (e.g. see [5, 16, 20, 24, 27, 29]).

Another approach to multi-paradigmatic modelling is reported by Lochmann and Hessellund [18]. They give a schema for working with multiple modelling languages where the first step is to identify the different domains and their connections. Then the connections are specified and finally the domains and the connections are implemented. In this way the different languages do not need to be combined on the

level of metamodels. A challenge for Lochmann and Hessellund is to ensure referential integrity across the connections and domains. Hessellund *et al.* [15] have developed a tool that was used for this purpose in an industrial project, SmartEMF. A similar approach is taken by Nentwich *et al.* [21], Denton *et al.* [9] and Warmer and Kleppe [30].

Motorola has applied model-driven engineering to describe the asynchronous message passing in a telecommunication system [7]. They discuss the challenges when different views of the system are integrated and how these can be overcome by using an aspect-oriented approach [11]. The Motorola authors implemented the signal processing by hand-written code.

3. MOTIVATION

In order to implement the LTE-A uplink channel estimator we decided to use two different modelling languages; a functional language to model the algorithms of the signal processing and an object-oriented language to model the execution flow of the signals. The different paradigms of the modelling languages will, ideally, ensure that the platform-independent models are closer to the requirements and domain descriptions than in the case of using one general modelling language [20]. In this way the functionality and structure of the domain implementations are to be free of platform-specific intrusions.

The setup poses two challenges:

1. To which extent is it possible to develop the signal processing and the flow of execution independently of each other?

2. How can the sub-solutions be integrated into one application running on the designated hardware?

Before we explore the challenges further we need to further define what we mean by a domain.

4. BACKGROUND

After defining what we mean by domain we describe the two domains in the LTE-A uplink channel estimator, the signal processing domain and the control domain.

4.1 Domain Definition

In our context a domain represents one subject matter with a set of well-defined concepts and characteristics [25] that cooperate to fulfill the interactions of the domain [23]. This definition of a domain is in line with what Giese *et al.* [12] call a horizontal decomposition and ensures the separation of concerns between the domains [10] as well as information hiding [22]. Furthermore, each domain can be realised as one or more software components [23].

4.2 The Signal Processing Domain

4.2.1 Signal Processing

What is often referred to within Ericsson AB as the signal processing domain, is characterized by a data centralized processing flow, where program state changes and external interactions are kept at a minimum, while more or less fixed and carefully optimized algorithms filter, convert or otherwise calculate on incoming data in a pre-deterministic way. In telecommunication applications, signal processing plays a crucial role, and the necessary algorithms have to be efficient in order to achieve the performance required on speed and quality.

$$\text{DCT-2}_n = \left[\cos \frac{k(2l+1)\pi}{2n}\right]_{0 \le k,l < n}$$

```
dct2 :: DVector Float -> DVector Float
dct2 xn = mat ** xn
  where mat =
  indexedMat (length xn) (length xn)
    (\k l -> dct2nkl (length xn) k l)

dct2nkl n k l =
  cos ((k'*(2*l'+1)*3.14)/(2*n'))
    where (n',k',l') = (intToFloat n,
                        intToFloat k,
                        intToFloat l)
```

Figure 1: The Discrete Cosine Transform matrix in mathematical and Feldspar notation.

In the LTE-A uplink testbed project, the signal processing more precisely consisted of multiple-user, multiple- antenna uplink data processing according to the 3GPP[1] standard, and our modelling mainly implicated the channel estimation parts.

4.2.2 Implementing Signal Processing

Today, using general high level languages, such as C or similar, there is often a large gap between the algorithm design and the implementation of the same. Due to this gap, the implementation of the signal processing algorithms can be indirect and unnecessarily complicated, and therefore error prone.

Feldspar is a domain-specific language currently developed by Chalmers University of Technology and Ericsson for signal processing [2]. The purpose is to limit the gap between the mathematical notation used in the design of signal processing algorithms and their implementation by using a functional modelling paradigm. Feldspar is embedded in Haskell[2], a third generation functional programming language, and tries to remain true to the Haskell syntax [3]. One important aspect of Feldspar is that it has no side-effects.

A Feldspar program can be evaluated directly via a Haskell interpreter, or be transformed into C by the accompanying code generator. In Figure 1 there is an example of a mathematical matrix multiplication used in signal processing together with the equivalent Feldspar definition, taken from Axelsson *et al.* [2].

4.3 The Control Domain

4.3.1 Controlling the Flow of Execution

We define the term Control Domain as a part of a software application controlling the flow of execution; responding to external communication and perhaps governed by internal state machinery. The control domain itself does not contain any complicated algorithmic complexity, instead it controls the order in which things are executed; in our case receiving and sending signals, initiating signal processing routines,

[1] `http://www.3gpp.org/specifications/`
[2] `http://www.haskell.org/`

and collecting their results.

4.3.2 Executable and Translatable UML

Our previous experiences at Ericsson show that an object-oriented modelling language is well suited for implementing the solutions needed for the control domain; modelling the interaction with surrounding applications in the system and managing the control and exchange of data between the different parts of the signal processing domain, regardless of implementation language chosen for those parts. Similar experiences from the telecommunications industry are reported on by Weigert and Weil [31].

Executable and Translatable UML (xtUML; [19, 23, 28]) evolved from merging the Shlaer-Mellor method [25] with the Unified Modeling Language (UML[3]).

xtUML has three kinds of diagrams, together with a textual action language. The diagrams are component diagrams, class diagrams and state machines. There is a clear hierarchical structure between the different diagrams; state machines are only found within classes, and classes are only found within components. Component diagrams have more or less the same syntax as in UML, but both class diagrams and state machines are more restricted in their syntax in comparison to UML. The action language is integrated with the graphical diagrams by the shared metamodel. At the time of our project the metamodel was propriety with restrictions on how it could be extended. The number of constructions in the metamodel is deliberately kept small so that there is always an appropriate correspondence in the platform-specific model. This also makes it an unsuitable language for complex algorithms since it has a very limited set of datastructures and only fundamental mathematical notations.

Since xtUML models have unambiguous semantics validation can be performed within the xtUML tool by an interpreter. During execution all changes of the association instances, attribute values and class instances are shown [17], as well as the change of state for classes with state machines, in the object model.

4.4 The Interface between the Domains

The actual processing of incoming data was - however static regarding the contents and order of algorithms involved - completely dynamic depending on the current configuration. In one instant, a certain configuration would put emphasis on a specific algorithm, quickly changing with a new configuration the next millisecond. This called for the need for independent possibilities for parallelization of each of the seven main algorithms, in order to continuously keep the processing latency at a minimum. This dynamicity in the control of invocation of algorithms decided in great extent the interfaces between the two identified domains.

5. CASE STUDY

5.1 Context

The application chosen for our case study was part of a larger Ericsson testbed project, already involving legacy and new software and hardware, and also new features. The testbed mainly involved 4G telecommunication baseband functionality, based on Ericsson's existing LTE products, both base station parts and user equipment parts. The testbed included adding features as well as deploying applications on a new hardware platform, resulting in an LTE-A [8] prototype to be presented at the Mobile World Congress in Barcelona, February 2011. Due to compiler availability for the new hardware the model transformations were restricted to generate C code.

The testbed project lasted for over one year. Already from the beginning it was emphasized that delivering an application that fulfilled the requirements within the calender deadline was more important than using specific methods and languages. There were no intentions of reusing anything of what the modelling could bring, except perhaps from the experienced gained.

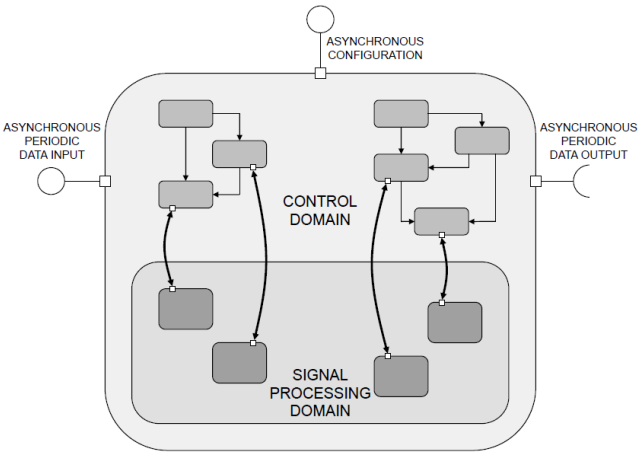

Figure 2: A schematic representation of how the state machines in the control domain cooperate with the independent algorithms of the signal processing domain.

5.2 Domain Identification

The application considered in this project was identified as a self-contained software component, managing a specific part of the data flow in an LTE-A base station, see Figure 2. The component's external interfaces were well specified, both in terms of parameter interchange and real-time responsibilities. The application was to be periodically provided with incoming data, while independently requested to update its configuration regarding how to process the data. Upon external triggering in one of these ways the control domain initiated internal chains of signal processing which were expected to run to completion.

5.2.1 Modelling Signal Processing using Feldspar

It was decided that it would be a good opportunity to test Feldspar by modelling the signal processing. There were other options, such as MatLab[4], but we wanted to take the opportunity to evaluate Feldspar's capability for modelling signal processing in a sharp project.

5.2.2 Modelling Control using xtUML

Ericsson has previously had success in using xtUML for reuse of platform-independent models [1] and test generation [6] while still finding the tool easy enough to use by

[3] http://www.uml.org/

[4] http://www.mathworks.se/products/matlab/

novice modellers [4]. We also know from experience that xtUML integrates nicely with legacy code written in C. Since the models are executable it is possible for users to validate that the models have the required functionality without generating code for deployment. Ericsson also had an in-house xtUML-to-C transformer that could compete with hand-written code [26]. And it was decided that one new modelling language was enough considering the firm deadline of the project. BridgePoint[5] was chosen as tool for modelling xtUML.

5.2.3 Modelling the Interface between the Domains

The interface between the two domains was defined in an implementation language independent way, identifying parameters necessary for describing the algorithmic and parallelization needs of the different algorithms within the data processing chain. Lochmann and Hessellund refer to such an implementation-independent interface as semantic [18]. The functionality of the interface was analysed using activity diagrams which was then manually transformed into C code.

5.3 Developers

The people involved in the project had been working between 5 to 15 years each with layer one baseband signal processing in telecommunication equipment at Ericsson. Mainly three developers were involved in the implementation of the models, two implementing the signal processing domain using Feldspar and one developer using xtUML for implementing the control domain. The developers had an unusual combination of expertise in that they were both domain experts and proficient C coders. In addition to the implementors, there was one domain expert in designing signal processing algorithms linked to the project as well as two experts in model transformations; one transformation expert for each modelling language. These two also served as mentors in respective modelling language. Two academic researchers participated as observers throughout the whole project.

The domain experts had chosen to participate in the project themselves. They had also chosen the modelling languages to use for implementing their domains. Their knowledge of C was a key reason for attempting to use a new modelling language, if Feldspar did not deliver it would always be possible to revert to the old ways and implement the signals using C. The motivation and commitment of the developers towards the project was one of the reasons that the project delivered a successful application.

5.4 Operation

The operation of the project was done in a Scrum way[6], using a backlog and in each sprint there were daily meetings and a burn down chart. After the end of each sprint there was a sprint review and delivery.

The testbed was horizontally decomposed [12] into a control and a signal processing domain according to the domain identification. The control domain was viewed as one autonomous component while the algorithms were developed as components of their own residing inside the control component. The implementation was done following a component-based development principle [13]. Our choice of language for each domain enabled testing of each software component independently and continuously throughout the implementation. When the implementation was complete the models were transformed into source code through code generation.

[5] `http://www.mentor.com/products/sm/model_development/bridgepoint/`

[6] `http://www.scrum.org/scrumguides/`

6. RESULTS

From the outcome of the case study we can now answer the challenges in section 3 in the order they were specified; first, to which extent is it possible to independently implement the domains? and secondly, how can the sub-solutions be integrated and deployed to constitute a running application?

6.1 Independent Implementation and Validation of the Domains

Working independently in the two modelling languages went well. The Feldspar models for the algorithms in the signal processing domain were implemented independently. Since the signal processing consisted of discretely implemented algorithms, they could be verified independently, processing premade input data, and comparing the output with likewise premade output data using the Haskell interpreter.

Simultaneously and independently, the xtUML models for the control domain could be implemented and tested. For the places where functionality defined in Feldspar were to be called from xtUML, stubs defined using action language were used within the executable control domain model, in order to verify the complete control flow; albeit not in a complete real-time aspect.

The designers implementing in Feldspar found it suitable for writing mathematical expressions, although it took some time to get used to the Feldspar syntax. There was initially only limited support for fixed-point arithmetics which is an important notion within the domain and the Haskell syntax for representing state was not intuitive compared to the mathematical formulation. The mapping from the mathematical notation of one of the algorithms into Feldspar code was done by the Feldspar mentor since it was not possible for the domain experts to do.

Using xtUML was straight-forward when modelling the solution for the control domain. Activity diagrams would have been preferable, since they would better describe the parallel execution of the signals than the state machines. The solution in our case was to model the control domain basic structure as activity diagrams outside the xtUML tool, in order to understand how to translate the behavior using the state machine diagrams provided by xtUML.

6.2 Integrating the Domain Solutions

Defining the interfaces required iteration over several meetings where the interfaces had to be refined and adapted due to extensions and changes on the functionality in the requirements. Since the understanding of the requirements on the interface increased during the implementation of the separate domains. These refinements were handled through the agile principles[7] inherent in the Scrum development process.

Testing the full interaction between the Feldspar and xtUML models required that they first were transformed into C code. The generated code was then validated before being deployed on the delivered hardware. Due to performance

[7] `http://agilemanifesto.org/principles.html`

limitations some hand-written C code with added intrinsics was necessary when deploying the generated code to maximise the usage of the platform-specific properties. Under the time constraints of the project this was quicker than updating the existing transformations [14].

7. DISCUSSION

7.1 The Right Language for the Right Task

The Feldspar developers needed time to get used to writing Feldspar code due to the design of the language. The authors of Feldspar claim that it is a domain-specific language for signal processing. Often, when developing a domain-specific language, one starts with the syntax used by the domain experts, which in our case would have been mathematical expressions for signal processing. This was not the case for Feldspar, since it is deliberately defined to have as similar syntax as possible as Haskell. Even though the gap between signal processing algorithms and Feldspar is a lot smaller than between signal processing algorithms and C there is still a gap. Moreover, Feldspar was a new language to learn for the domain experts and it belongs to a different programing paradigm than C which they were used to. The problem was limited by the fact that one of the people working in the Feldspar project was part of the group during the initial phase of the project.

From our experience of xtUML [4] it does not take that long time to get used to the syntax of the modelling elements, as it does to get used to the tool that creates them. More importantly, as the project evolved we saw a need for handling the data flow in a way that neither xtUML nor the existing platform supported.

7.2 Multicore

At the time of our project there was no way to generate efficient code for multicore deployment, neither from Feldspar nor from xtUML, so we opted to implement the interface in C straight away. However, both languages chosen for its respective domain suggested suitable inherent properties for such a transformation to be plausible in many aspects. The feature of Feldspar functions having no side effects, together with designing the algorithms as a library of dynamically composable low-level operations would be well suited for execution in a distributed and concurrent manner. And implementing dynamically created independent instances of state machinery in xtUML would only benefit from being run as distributed and concurrent threads in a multicore environment.

Exploiting properties already naturally inherent within the domain languages, when transforming the domain models into generated code, would therefore render it possible to generate code suitable for multicore deployment - for free, so to speak. Naturally it would require considerately more development by the transformation experts to customise the transformations in order to obtain optimal performance for a specific platform.

8. CONCLUSION AND FUTURE WORK

In relation to previous work we chose not to unify the metamodels of the modelling languages since we did not have straight access to the xtUML metamodel. Instead we relied on the identification of a logical interface between the different domains and modelling languages. In this aspect our work is related to the findings reported by Lochmann and Hessellund [18]. By adhering to an agile and component-based development strategy [13] and through the separation of concerns [10] between the domains, the referential integrity is ensured through the interface between the domains [23]. In comparison to the work at Motorola [7, 31] we have used an in-house modelling language to implement the signal processing.

The development process relied on the possibility to validate the domains independently. For this to be possible it is important that the modelling languages are executable in their own right and that they later on can be translated into deployable code, after the models have been validated to have the right structure and behaviour [23].

We believe that multicore is both a challenge and an opportunity for model-driven software development. Multicore is a complex paradigm of its own and if the platform-independent properties of software modelling could be combined with efficient code generation a lot would be won.

9. REFERENCES

[1] S. Andersson and T. Siljamäki. Proof of Concept - Reuse of PIM, Experience Report. In *SPLST'09 & NW-MODE'09: Proceedings of 11th Symposium on Programming Languages and Software Tools and 7th Nordic Workshop on Model Driven Software Engineering*, Tampere, Finland, August 2009.

[2] E. Axelsson, K. Claessen, G. Dévai, Z. Horváth, K. Keijzer, B. Lyckegård, A. Persson, M. Sheeran, J. Svenningsson, and A. Vajdax. Feldspar: A domain specific language for digital signal processing algorithms. In *Formal Methods and Models for Codesign (MEMOCODE), 2010 8th IEEE/ACM International Conference on*, pages 169 –178, July 2010.

[3] E. Axelsson, K. Claessen, M. Sheeran, J. Svenningsson, D. Engdal, and A. Persson. The Design and Implementation of Feldspar – An Embedded Language for Digital Signal Processing. In *Proceedings of the 22nd international conference on Implementation and application of functional languages*, IFL'10, pages 121–136, Berlin, Heidelberg, 2011. Springer-Verlag.

[4] H. Burden, R. Heldal, and T. Siljamäki. Executable and Translatable UML – How Difficult Can it Be? In *APSEC 2011: 18th Asia-Pacific Software Engineering Conference*, Ho Chi Minh City, Vietnam, December 2011.

[5] S. Burmester, H. Giese, J. Niere, M. Tichy, J. P. Wadsack, R. Wagner, L. Wendehals, and A. Zündorf. Tool integration at the meta-model level: the Fujaba approach. *International Journal on Software Tools for Technology Transfer (STTT)*, 6:203–218, 2004.

[6] F. Ciccozzi, A. Cicchetti, T. Siljamäki, and J. Kavadiya. Automating test cases generation: From xtUML system models to QML test models. In

MOMPES: Model-based Methodologies for Pervasive and Embedded Software, Antwerpen, Belgium, September 2010.

[7] T. Cottenier, A. van den Berg, and T. Elrad. Motorola WEAVR: Aspect and Model-Driven Engineering. *Journal of Object Technology*, 6(7):51–88, August 2007. Aspect-Oriented Modeling.

[8] E. Dahlman, S. Parkvall, and J. Sköld. *4G: LTE/LTE-Advanced for Mobile Broadband*. Academic Press. Elsevier/Academic Press, 2011.

[9] T. Denton, E. Jones, S. Srinivasan, K. Owens, and R. W. Buskens. NAOMI — An Experimental Platform for Multi—modeling. In *Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems*, MoDELS '08, pages 143–157, Berlin, Heidelberg, 2008. Springer-Verlag.

[10] E. W. Dijkstra. EWD 447: On the role of scientific thought. *Selected Writings on Computing: A Personal Perspective*, pages 60–66, 1982.

[11] R. Filman, T. Elrad, S. Clarke, and M. Aksit. *Aspect-Oriented Software Development*. Addison-Wesley Professional, first edition, 2004.

[12] H. Giese, S. Neumann, O. Niggemann, and B. Schätz. Model-Based Integration. In H. Giese, G. Karsai, E. Lee, B. Rumpe, and B. Schätz, editors, *Model-Based Engineering of Embedded Real-Time Systems*, volume 6100 of *Lecture Notes in Computer Science*, chapter 2, pages 17–54. Springer Berlin/Heidelberg, 2011.

[13] G. T. Heineman and W. T. Councill, editors. *Component-based software engineering: putting the pieces together*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[14] R. Heldal, H. Burden, and M. Lundqvist. Limits of Model Transformations for Embedded Software. In *35th Annual IEEE Software Engineering Workshop*, Heraklion, Greece, October 2012. IEEE.

[15] A. Hessellund, K. Czarnecki, and A. Wąsowski. Guided Development with Multiple Domain-Specific Languages. In G. Engels, B. Opdyke, D. Schmidt, and F. Weil, editors, *MoDELS'07: Model Driven Engineering Languages and Systems*, volume 4735 of *Lecture Notes in Computer Science*, pages 46–60. Springer Berlin / Heidelberg, 2007.

[16] G. Kainz, C. Buckl, S. Sommer, and A. Knoll. Model-to-Metamodel Transformation for the Development of Component-Based Systems. In D. Petriu, N. Rouquette, and y. Haugen, editors, *Model Driven Engineering Languages and Systems*, volume 6395 of *Lecture Notes in Computer Science*, pages 391–405. Springer Berlin / Heidelberg, 2010.

[17] C. Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[18] H. Lochmann and A. Hessellund. An Integrated View on Modeling with Multiple Domain-Specific Languages. In *Proceedings of the IASTED International Conference on Software Engineering*, pages 1–10. ACTA Press, February 2009.

[19] S. J. Mellor and M. Balcer. *Executable UML: A Foundation for Model-Driven Architectures*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[20] P. J. Mosterman and H. Vangheluwe. Computer Automated Multi-Paradigm Modeling: An Introduction. *SIMULATION: The Society for Modeling and Simulation International*, 80(9):433–450, September 2004.

[21] C. Nentwich, W. Emmerich, and A. Finkelstein. Consistency management with repair actions. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 455–464, Washington, DC, USA, 2003. IEEE Computer Society.

[22] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, December 1972.

[23] C. Raistrick, P. Francis, J. Wright, C. Carter, and I. Wilkie. *Model Driven Architecture with Executable UML^{TM}*. Cambridge University Press, New York, NY, USA, 2004.

[24] C. Rodríguez, M. Sánchez, and J. Villalobos. Metamodel Dependencies for Executable Models. In J. Bishop and A. Vallecillo, editors, *TOOLS (49)*, volume 6705 of *Lecture Notes in Computer Science*, pages 83–98. Springer, 2011.

[25] S. Shlaer and S. J. Mellor. *Object lifecycles: modeling the world in states*. Yourdon Press, Upper Saddle River, NJ, USA, 1992.

[26] T. Siljamäki and S. Andersson. Performance Benchmarking of real time critical function using BridgePoint xtUML. In *NW-MoDE'08: Nordic Workshop on Model Driven Engineering*, Reykjavik, Iceland, August 2008.

[27] J. Sprinkle, B. Rumpe, H. Vangheluwe, and G. Karsai. Metamodelling. In H. Giese, G. Karsai, E. Lee, B. Rumpe, and B. Schätz, editors, *Model-Based Engineering of Embedded Real-Time Systems*, volume 6100 of *Lecture Notes in Computer Science*, chapter 3, pages 57–76. Springer Berlin/Heidelberg, 2011.

[28] L. Starr. *Executable UML: How to Build Class Models*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[29] A. Vallecillo. On the Combination of Domain Specific Modeling Languages. In T. Kühne, B. Selic, M.-P. Gervais, and F. Terrier, editors, *ECMFA*, volume 6138 of *Lecture Notes in Computer Science*, pages 305–320. Springer, 2010.

[30] J. Warmer and A. Kleppe. Building a Flexible Software Factory Using Partial Domain Specific Models. In *Sixth OOPSLA Workshop on Domain-Specific Modeling (DSM'06)*, pages 15–22, Jyvaskyla, October 2006. University of Jyvaskyla.

[31] T. Weigert and F. Weil. Practical Experiences in Using Model-Driven Engineering to Develop Trustworthy Computing Systems. *Sensor Networks, Ubiquitous, and Trustworthy Computing, International Conference on*, 1:208–217, 2006.