Limits of Model Transformations for Embedded Software

Rogardt Heldal
Computer Science and Engineering
Chalmers University of Technology
and University of Gothenburg
Gothenburg, Sweden
Email: heldal@chalmers.se

Håkan Burden
Computer Science and Engineering
Chalmers University of Technology
and University of Gothenburg
Gothenburg, Sweden
Email: burden@chalmers.se

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

***Abstract*—We report on an effort to use executable modeling languages for developing software for the Ericsson LTE-A uplink test bed. The test bed was a part of a 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 existing model transformations were not efficient enough on the new platform and that model transformations across paradigms is a challenging task; to meet the deadline and the requirements the generated code had to be manually optimised.**

I. Introduction

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

An answer to this problem is Model-Driven Architecture (MDA; [1], [2]). In MDA the functionality of the system is encoded in platform-independent models that abstract away from hardware- and programming language-specific issues. This way the models become reusable assets across platforms. Besides describing the system, the models can be used to generate a platform-specific implementation through model transformations.

Contribution: We show two challenges for model transformations within the telecom industry; transformations from a functional to an imperative paradigm and reuse of existing transformations for new hardware.

Overview: In section II we will further motivate our case study. In section III we give the necessary background about the investigated domains and what we consider suitable modeling languages for each domain. In section IV the delivered application is described together with the implementation process. The outcome of the process is then found in section V which is followed by our discussion in section VI. We relate our own findings to previous work in section VII before we conclude in section VIII.

II. Motivation

To what extent are model transformations reusable? According to Mellor *et al.* [2] the vision of MDA is to have reusable transformations for new deployments. On the other hand the MDA Guide [1] delivered by the Object Management Group[1] states that transformations might need adjusting for new deployments. In addition to this conflict among MDA promoters, Mohagheghi and Haugen [3] state that the efficiency and completeness of the generated code are two important aspects when using models in software development. To further investigate the nature of model transformations we evaluate two modeling languages, one functional and one object-oriented, with regards to their code generation capabilities during a project at Ericsson.

The task was to deliver a subsystem of an Ericsson test bed for a 4G telecommunications system, namely the uplink part of an LTE-A radio base station [4]. The requirements on such an application include unconditional real-time performance for calculations on synchronous data and the contextual determination of when signals shall be sent and processed as well as operational reliability on new hardware. Since the system was to be demonstrated at the Mobile World Congress in Barcelona, February 2011, there was a firm dead-line. In this context, hand-written C (or sometimes even Assembly) is the current state-of-the-art when it comes to optimally exploiting the hardware's processors and memory. For platform-independent models to provide the software implementations, it is necessary to be able to generate code from them that performance-wise is comparable to hand-written C code [5].

The calculations of the data were modeled using a new functional modeling language with efficient generation of C

[1] http://www.omg.org/

code, developed in-house. To model the control flow we opted to use an object-oriented modeling language with proven code generation capabilities. Given the requirements on real-time performance and the firm dead-line this raised two questions:

Research Question 1: Which are the key challenges to consider when generating imperative code from a functional modeling language?

Research Question 2: To what extent is it possible to reuse existing model transformations for new hardware?

III. Background

Before we turn our attention towards the case study we will first define the domains and modelling languages that were considered.

A. Domain Definition

For us a domain represents one subject matter which is an autonomous world with a set of well-defined concepts and characteristics [6] that cooperate to fulfill a well-defined interface [7]. This definition of a domain is in line with what Giese *et al.* [8] call a horizontal decomposition and ensures the separation of concerns between the domains [9] as well as information hiding [10]. Each domain can be realised as one or more software components as long as these are described by the same platform-independent modeling language [7].

B. The Signal Processing Domain

1) Signal Processing: Within Ericsson the signal processing domain is characterized by a data 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 incoming data. 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.

2) Implementing Signal Processing: In order to obtain necessary optimized performance on the intended deployment platform, the implementation of the signal processing has to be padded with non-standardized, hardware specific instructions. These instructions are referred to as intrinsic functions, and specify how the specific hardware should be used, in a stricter way than just by compiling standard C or similar. Although the signal processing solution may be fixed and thoroughly verified, every change in deployment platform will bring on the need for new manual intervention, since optimization might be reached in new ways, using other code styles and intrinsic functions.

Feldspar is a domain-specific language currently developed by Chalmers University of Technology and Ericsson for signal processing [11]. The purpose is to limit the gap between the mathematical notation used in the design of signal processing algorithms and their implementation. Feldspar is embedded in Haskell[2] and has no side-effects. A Feldspar program can be evaluated directly via a Haskell interpreter. There is also a code generator that transforms Feldspar programs into C, since run-time performance plays such an important role within signal processing. In Fig. 1 there is an example of a mathematical matrix multiplication used in signal processing together with the equivalent Feldspar definition [11].

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

```
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)
```

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

C. The Control Domain

1) Controlling the Flow of Execution: We define the term Control Domain as a part of a software application controlling the flow of execution through internal state machinery responding to external communication. 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, and collecting their results.

2) Executable and Translatable UML: Previous experiences show that an object-oriented modeling language is well suited for describing the control domain; modeling 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 [12].

Executable and Translatable UML (xtUML; [7], [13], [14]) evolved from merging the Shlaer-Mellor method [6] 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. There is an action language, integrated with the graphical elements by a shared meta model [14]. The number of constructions 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 limited set of data structures and only fundamental mathematical notations.

Since xtUML models have unambiguous semantics validation can be performed within the xtUML model by an

[2]http://www.haskell.org/

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

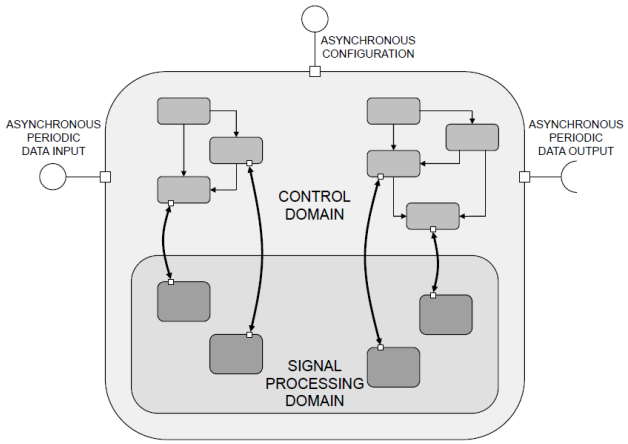

Fig. 2. The considered application consisted of the control domain, in turn enclosing the signal processing domain.

interpreter. During execution all changes of the association instances, attribute values and class instance are shown [15] as well as the change of state for classes with state machines in the object model.

IV. Case Study

A. Context

The application chosen for our case study was part of a larger Ericsson test bed project, already involving legacy and new software and hardware, and also new features. The test bed involved 4G telecommunication baseband functionality, based on Ericssons existing LTE products, both base station parts and user equipment parts. The test bed included adding features as well as deploying applications on a new hardware platform, resulting in an LTE-A [4] prototype to be presented at the Mobile World Congress in Barcelona, February 2011.

The test bed project lasted part-time for over one year. Already from the beginning it was emphasized that delivering an application that fulfilled the requirements within deadline was more important than using specific methods or languages.

B. 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 the LTE-A base station, see Fig. 2. The components 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 incoming data. Upon external triggering in one of these ways, internal chains of data processing was initiated, and expected to run to completion.

1) Modeling Control using xtUML: Previous experiences with xtUML at Ericsson include reuse of platform-independent models [12] and test generation [16] while still finding the tool easy enough to use by novice modellers [17]. We also knew from experience that xtUML integrates nicely with legacy code written in C.

Since the models are executable it is possible for designers to validate that the models have the required functionality without generating code for deployment. Finally, Ericsson had an existing xtUML-to-C transformation that could compete with hand-written code [5]. We chose BridgePoint[4] as the tool for modeling xtUML.

2) Modeling Signal Processing using Feldspar: It was decided that it would be a good opportunity to test Feldspar for modeling the signal processing. Feldspar had never been used in an industrial development project before and we wanted to see how well it would fulfill our requirements on real-time performance for the generated code.

3) Modeling the Interface between the Domains: The actual processing of incoming data was 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. Since the number of signal processes and available processors changed from one millisecond to the next it called for the need for independent possibilities for parallelization of each of the involved algorithms, in order to continuously maximize the processing throughput, while keeping the processing latency at a minimum.

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 interface as semantic [18].

C. 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 modeling language. These two also served as mentors in respective modeling language. Two academic researchers participated as observers and participated in Ericsson AB's internal discussions regarding modelling and model transformations.

D. Operation

The operation of the project was inspired by the Scrum method[5], using a backlog and in each sprint there were daily meetings and a burn down chart. At the end of each sprint there was a sprint review and a delivery.

[4]http://www.mentor.com/products/sm/model_development/bridgepoint/
[5]http://www.scrum.org/scrumguides/

The implementation was done following a component-based development principle [19], where the control domain was viewed as one autonomous component and each algorithm as a component of its own residing inside the control component. 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 target code.

V. Results

Neither the Feldspar nor the xtUML transformation met the requirements of the project and they were both too complicated for those developers without transformation expertise to change.

A. Transformations across paradigms

Since memory was sparse on the designated platform all variables were limited to 16 bits representation. When transforming Feldspar into C we encountered a problem; the transformation introduced an internal variable for storing an intermediate result. This variable required better precision than the memory constraint allowed, in order to get good enough results 32 bits representation was needed for its encoding. Since this variable was not present in the Feldspar code it could not be marked [2] for exclusive treatment and if all variables were given more memory it would overflow the platform. Due to the time limit the only possibility was to manually change the generated code so that one particular variable accessed an increased memory space.

B. Reuse of Transformations

At a crucial moment in the project the xtUML transformation expert was moved to another project taking place in another country. This meant that there was not enough time to change the organization of the project to adapt the existing transformation to the new platform. This adaptation was necessary since the generated C code could not obtain the appropriate utilization of the limited memory and processing capabilities to meet the requirements on real-time performance.

C. Transforming the Interface to Multicore

Neither of the existing transformations could adequately handle the multicore parallelization of the platform. This came as no surprise since both lacked a way of specifying the necessary dynamic configurations. The behavior and structure when realizing the interface deployment was therefore coded by hand.

VI. Discussion

The possibility for reusing existing solutions is supposed to be one of the strengths of using a model-driven approach to software development [2]. Our case study indicates that this was not the case, at least not when the transformations themselves need to be updated, as was the case in our project. We believe this is particularly true when a new platform is involved. Maybe after a few projects are completed for the same platform, all the relevant platform-specific knowledge is encoded in the transformation. In industrial settings where hardware is continuously changed and updated this is going to be a challenge that MDA has to address for each modeling language that is used within a project.

Hutchinson *et al.* [20] argue that there are too few who have the necessary skills in developing domain-specific languages and efficient transformations within industry. A key issue for successful MDA in industry is the access to transformation experts with short notice since it is not always possible to foresee when a transformation needs to be updated or optimized. We believe that the importance of the transformations will increase as the number of modeling languages in a project grows. We have probably only started to see the issue of transformation reuse in industry; Lettner *et al.* [21] also report on the problems of porting existing solutions to new platforms in their case study. This is in contrast to the view taken by Kelly and Tolvanen [22] who claim that defining your own modelling language with transformation to code is more efficient than using a general purpose language such as C.

We found that it is vital for an MDE project to have access to the model transformation developers when needed since it is not possible to foresee when a transformation is not going to meet the combined requirements of the project and new hardware. For each modeling language used within a project containing code generation there is going to be one more transformation to adapt and optimize for.

Even if efficient code generation for multicore was not attainable we see some promising possibilities in the inherent properties of the chosen languages. Feldspar functions have no side effects and designed as a library of dynamically composable low-level operations, they would be well suited for execution in a distributed and concurrent manner. Independent instances of state machinery in xtUML can also be deployed as distributed and concurrent threads in a multicore environment.

VII. Related work

Hutchinson *et al.* [20] give a general overview of the current industrial practice in model-based software development. As seen in the Discussion, section VI, our experiences relate to their findings on the importance of transformation experts in MDA projects.

Motorola has applied model-driven engineering to describe the asynchronous message passing in a telecommunication system [23]. They also split their system into domains by hierachical decomposition. The difference lies in that while we have used a modeling language to describe the signal processing they have used hand-written code. Another report from Motorola, by Weigert and Weil [24], also found that the transformations had to be adapted to fit new hardware when porting their models. Just as for the previous Motorola report, they do not use models to implement the complex algorithmic behaviour.

VIII. Conclusion and Future work

The project was a success in terms of meeting deadline and performance, but that was largely due to the fact that the

model implementers also had good knowledge of C; the target language of the model transformations. Based on this project we cannot recommend anyone to commence on a similar project without knowing the target language and platform well when considering real time systems running on new platforms. This is due to the fact that the quality of the target code is highly important for performance and it might not be as easy as expected to reuse existing models and transformations.

In our point of view if industry shall succeed in using MDA more research is needed on model transformations, both from an organisational view and the perspective of the efficiency of the generated code. Today, there are usually only a few gurus within companies who have the necessary competence in model transformation [20]. MDA projects will then get too dependent on these transformation experts which creates a bottleneck if they are not continuously accessible during development. Whittle and Hutchinson report on the necessity of educating more software modelers with transformation skills [25]. Based on our own experience we can only agree.

The challenge of generating code across paradigms is another area that needs further exploration. Not only in the case of moving from a functional paradigm to an imperative but also in the case of modeling for multicore and the subsequent transformation to many platforms.

REFERENCES

[1] J. Miller and J. Mukerji, "MDA Guide Version 1.0.1," Object Management Group (OMG), Tech. Rep., 2003.

[2] S. J. Mellor, S. Kendall, A. Uhl, and D. Weise, *MDA Distilled*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 2004.

[3] P. Mohagheghi and Ø. Haugen, "Evaluating Domain-Specific Modelling Solutions," in *ER Workshops*, ser. Lecture Notes in Computer Science, J. Trujillo, G. Dobbie, H. Kangassalo, S. Hartmann, M. Kirchberg, M. Rossi, I. Reinhartz-Berger, E. Zimányi, and F. Frasincar, Eds., vol. 6413. Springer, 2010, pp. 212–221.

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

[5] 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.

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

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

[8] H. Giese, S. Neumann, O. Niggemann, and B. Schätz, "Model-Based Integration," in *Model-Based Engineering of Embedded Real-Time Systems*, ser. Lecture Notes in Computer Science, H. Giese, G. Karsai, E. Lee, B. Rumpe, and B. Schätz, Eds. Springer Berlin/Heidelberg, 2011, vol. 6100, ch. 2, pp. 17–54.

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

[10] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Commun. ACM*, vol. 15, no. 12, pp. 1053–1058, December 1972.

[11] 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*, July 2010, pp. 169 –178.

[12] 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.

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

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

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

[16] 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.

[17] 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.

[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*. ACTA Press, February 2009, pp. 1–10.

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

[20] J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen, "Empirical assessment of MDE in industry," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: ACM, 2011, pp. 471–480.

[21] M. Lettner, M. Tschernuth, and R. Mayrhofer, "A Critical Review of Applied MDA for Embedded Devices: Identification of Problem Classes and Discussing Porting Efforts in Practice," in *Model Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science, J. Whittle, T. Clark, and T. Kühne, Eds. Springer Berlin / Heidelberg, 2011, vol. 6981, pp. 228–242.

[22] S. Kelly and J.-P. Tolvanen, *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Pr, March 2008.

[23] T. Cottenier, A. van den Berg, and T. Elrad, "Motorola WEAVR: Aspect and Model-Driven Engineering," *Journal of Object Technology*, vol. 6, no. 7, pp. 51–88, August 2007, aspect-Oriented Modeling.

[24] 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*, vol. 1, pp. 208–217, 2006.

[25] J. Whittle and J. Hutchinson, "Mismatches between industry and teaching of model-driven software development," in *7th Educators' Symposium@MODELS 2011 – Software Modeling in Education*, M. Brandsteidl and A. Winter, Eds., Wellington, New Zealand, September 2011, pp. 27–30.