Model-based formal specification of a DSL library for a qualified code generator *

Arnaud Dieumegard
IRIT-ENSEEIHT, Université de Toulouse
2 rue Charles Camichel
Toulouse, France
adieumeg@enseeiht.fr

Andres Toom
Institute of Cybernetics at Tallinn University of Technology
Akadeemia tee 21
EE-12618 Tallinn, Estonia
andres@krates.ee

Marc Pantel
IRIT-ENSEEIHT, Université de Toulouse
2 rue Charles Camichel
Toulouse, France
marc.pantel@enseeiht.fr

ABSTRACT

Critical embedded systems development is a complex and highly sensitive task. Model-driven engineering (MDE) intends to bridge the gaps between the different parts of this process: high-level requirements, design, implementation and verification, by promoting formalization of the various process artefacts as models. This paper focuses on the rigorous and flexible model-based specification and implementation of a part of the requirement language of an embedded code generator. It relies on the use of OCL integrated in a textual specification language as a mean to formally specify graphical modeling languages such as SIMULINK and SCICOS and their extensible sophisticated block libraries.

Keywords

MDE, OCL, formal specification, software qualification, automatic code generation, SIMULINK, SCICOS

1. INTRODUCTION

Embedded and safety critical software is of strategic importance for the industry and society. Automatic code generation (ACG) is expected to prevent a large class of errors originating from the manual translation of system specifications to code. However, many industrial ACG-s are complex proprietary tools, whose adaptability to the customer needs is very limited. Development of safety critical systems is performed according to normative guidelines, such as IEC-61508, DO-178, ISO-26262, ECSS, but the verification activities involve still a large amount of proofreading and non-exhaustive testing. The recent release of DO-178C has given guidelines for combining formal and model oriented techniques with software certification.

2. GENEAUTO

The current work was started in the GENEAUTO[1] project, an open-source code generation toolset intended to be used and qualified according to the high-integrity domain quality and safety standards. The toolset takes as input software specification written in graphical dataflow-style modeling languages like SIMULINK and SCICOS or state-chart notation like STATEFLOW and produces C or Ada code as output. This work is being extended in the projects PROJECTP[2] and Hi-MoCo[3].

3. SPECIFYING SIMULINK AND SCICOS BLOCK LIBRARIES

An important part of these languages is an extensible block (function) library that contains common building blocks for different applications. These ones can by very complex as the implemented algorithm is sometimes complicated, but also because is has a high variability through different parameters like number and type of inputs, rounding and saturation options, etc. This contribution proposes a way to deal with this second source of complexity in the block specification.

In GENEAUTO, classical natural language specification was chosen for first prototyping and there was no direct link between the blocks specification and implementation. Verification was performed by proofreading and testing against manually developed unit tests. Although the mechanism for implementing block library itself was open and flexible, such a decoupled process is error-prone and not easy to maintain.

Mathematical MathML or LaTeX style specification was also investigated, but the source code of such representations was found to be too complex and hard to maintain.

A third, model-based approach is described next.

4. MODEL-BASED SPECIFICATION OF THE BLOCK LIBRARY

Modeling can be used both for crafting the requirements/ specifications, but also the implementation. We chose to design a model-based textual DSL relying on standard MOF and OCL concepts in order to formally define the structure of blocks and associated *design by contract* style formal specification of their semantics. This approach is more likely to be accepted by engineers and certification authorities, as it relies on common and standard technology.

We are using ECORE, a widely used variant of MOF, for which there exists a lot of tool support. The basic structure of our DSL is specified as an ECORE metamodel. OCL is

*This work has been partly funded by the ITEA2 project OPEES and by the French and Estonian Science Foundations throught the Parrot program.

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

[2] http://www.open-do.org/projects/p/

[3] http://www.eurekanetwork.org/project/-/id/6037

used for specifying some additional structural constraints, but more importantly, the computational semantics of each block in the library.

Due to the many configuration options in the block libraries for our input languages, we have adopted a *feature modeling* [4] like approach for our DSL. Figure 1 shows the BlockLibrary metamodel that includes the BlockMode and BlockVariant concepts. BlockVariant associates a set of related parameters that might affect the block's port signature. A BlockMode is a set of valid configurations called BlockVariant that have similar behavior from the semantic viewpoint. This structure, unlike the traditional tree structure used in feature modeling, forms a multiply rooted directed acyclic graph allowing reuse of specification elements and separation between structural and semantic constraints.

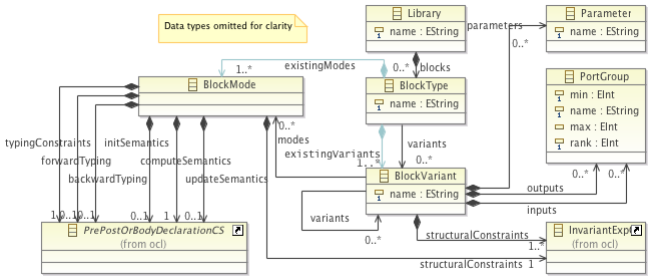

Figure 1: BlockLibrary specification metamodel

5. APPLICATIONS OF THE MODEL-BASED SPECIFICATION

The block specification language is meant to be used by the block library designers for giving a semantic reference and contract to its users. But its formal nature can also serve other purposes.

We have developed *textual editors* for our DSL integrating the BlockLibrary and OCL concrete syntaxes using XTEXT and EMFTEXT. Similarly, we have implemented *documentation generation* at little additional cost, using the ACCELEO tool.

Consistency and completeness of the specification of blocks is not ensured by the usage of a formal specification language alone. For more powerful verification it would be possible to transform the block library language and OCL constraints into inductive data types of a proof assistant input language. Alternatively, they could be transformed to other kinds of proof environments, such as [2].

In GENEAUTO code generation from library blocks relies on *typing* and *code generation* functions associated with the block definition. From a formal block library definition, the skeletons of such functions can be automatically generated and augmented with verification conditions that the implementation must satisfy. Furthermore, if these constraints are integrated in the generated code, it helps to verify the generated code using proof checkers or other static analysis tools. Such an extension has been implemented for GENEAUTO and has been applied in the successful verification of an helicopter horizontal stability control [7].

6. RELATED WORKS

Our work aims at providing a practical way for specifying formally a DSL library. Such specification must be easily maintainable and readable. We have used similar ideas as in the *feature modeling* [4] methodology to formalize the variable parts in block specifications. A similar approach has been followed in [6].

Many works on the verification of formal specifications and their usage in the development rely on complex specification languages that are not suited for common industry engineers. Most of them focus only on the verification of specifications like [3] or test case generation [1].

A close approach to ours is [5] on the use of an architecture description language for embedded hardware. It relies on a DSL for formal specification of the input hardware that is then used for the generation of tools compilers). These tools are in turn embedded in a critical system development.

7. CONCLUSION/FUTURE WORKS

We have presented in this paper a methodology for formally specifying a block library for modeling languages like SIMULINK and SCICOS. The difficulty of this task lays in the high number of parameters and the way they affect the blocks' semantics. We have designed a dedicated DSL that integrates the block library metamodel and the general OCL specification language. This language allows to write specifications that are usable for semantic reference, automatic handling of blocks by tools such as code generators and simulators, as well as contracts for the blocks' implementation. We plan to refine this language further and use it to fully specify the block library of the GENEAUTO successor tool developed in the project PROJECTP. Examples and resources related to this case study can be found at `http://dieumegard.perso.enseeiht.fr/blocklibrary`.

8. REFERENCES

[1] H. Ben-Abdallah, D. Clarke, I. Lee, and O. Sokolsky. PARAGON: A Paradigm for the Specification, Verification, and Testing of Real-Time Systems. In *IEEE Aerospace Conference*, 1997.

[2] D. Calegari, C. Luna, N. Szasz, and Á. Tasistro. A Type-Theoretic Framework for Certified Model Transformations. volume 6527 of *Lecture Notes in Computer Science*. 2011.

[3] C. Heitmeyer, J. Kirby, and B. Labaw. Tools for Formal Specification, Verification, and Validation of Requirements. In *COMPASS'97*.

[4] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie-Mellon University, 1990.

[5] G. Savaton, J.-L. Béchennec, M. Briday, and R. Kassem. An Architecture Description Language for Embedded Hadware Platforms. In *OCL and textual modelling 2011 (EASST)*.

[6] D. Streitferdt, M. Riebisch, and I. Philippow. Details of formalized relations in feature models using ocl. In *ECBS*, 2003.

[7] T. E. Wang, A. Dieumegard, E. Feron, R. Jobredeaux, M. Pantel, and P.-L. Garoche. Autocoding Of Computer-Controlled Systems With Semantics For Formal Code Verification. In *S5 Symposium*, 2012.