Model Based Development of Safety-Critical Systems Using Template Based Code Generation

Matthias Regensburger, Christian Buckl, Alois Knoll, Gerhard Schrott
Department of Informatics
Technische Universität München Garching b. München, Germany
{regensbu,buckl,knoll,schrott}@in.tum.de

Abstract

Model-based development is state of the art in software engineering, due to its potential regarding automatic code synthesis. Nevertheless for embedded systems, where there exists a huge heterogeneity of used platforms, it is obvious that it is impossible to design a code generator that supports a priori all required platforms. Instead a code generator architecture is needed that is suited for an easy extensibility of the code generation ability. One possible solution is the use of template-based approaches. In this paper, we describe an approach[1] to develop safety-critical real-time systems by using openArchitectureWare, a modular MDA/MDD generator framework. We will present the tool-chain and discuss two lab applications.

1 Introduction

Model-based development tools are only of limited value for the use in embedded systems, since they focus typically on the application functionality and do not cover system aspects like fault-tolerance mechanisms. Two main reasons can be identified, why such code is not targeted. Firstly, the used models are not suited for the generation of fault-tolerance mechanisms, due to the special requirements for this process. Such requirements are the necessity of explicit execution semantics and the support of replica determinism. The use of domain specific models[2] can solve this problem. One possible approach is described in [1].

The other problem is related to the platform dependability of code. The concrete implementation depends on the used hardware, operating system and programming language. It is obvious that it is impossible to implement a code generator that supports a priori all possible combinations. In addition, it must be possible to add new fault-tolerance mechanisms. Therefore, the code generator must be designed in a way that it can be easily extended, even by the user. In this paper, we will focus on one solution using template-based code generators. Instead of implementing a new code generator, we will point out the possibilities to use existing frameworks, e.g. openArchitectureWare (oAW)[2]. OpenArchitectureWare is a very powerful and configurable model-driven architecture (MDA) and model-driven development (MDD) suite for generating, transforming and checking models.
The paper is constructed as follows: the different steps that can be performed when using oAW are described in the following section. Subsequent, we will give some details about the realized lab applications, before we summarize the approach and point out future research directions.

2 Tool-Chain

Within this section of the paper, we will describe the use of openArchitectureWare (oAW) in our system. Figure 1 depicts the steps of the code generation process. Basis for the code generation are the models specified by the application developer. The modeling tool incorporated within oAW allows the specification of the models using graphical notations, see

[1]This work is funded by the German Federal Ministry of Education and Research BMBF under grant 01ISF12A

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

0-7695-3054-0/07 $25.00 © 2007 IEEE
DOI 10.1109/PRDC.2007.25

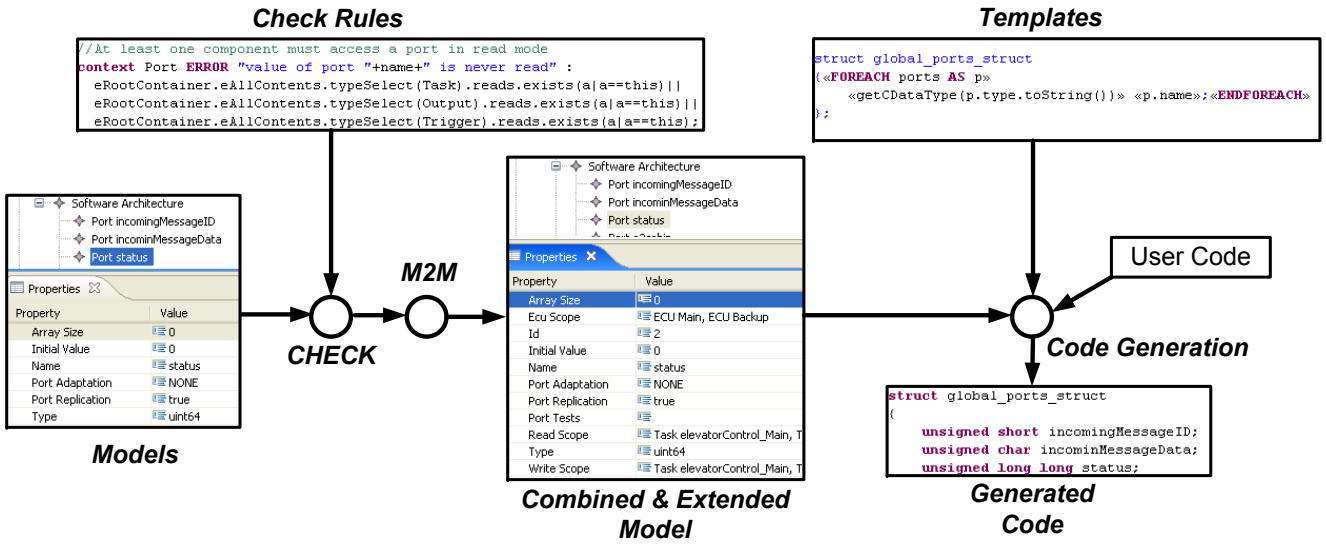

Figure 1. Code Generation Steps

Section 2.1. Therefore, syntactical errors are excluded by design. Nevertheless, it is necessary to check the semantical correctness of the models. This is done automatically by executing several tests, see Section 2.2. The validated models are then combined to one model. The combination includes also the calculation of extra information that can be used to simplify code generation. An example is given in Section 2.3. The extended and combined model is then used for code generation. Within this process, appropriate templates to solve application aspects are selected and adopted to the application, see Section 2.4. The result is a tailored runtime system, including mechanisms for scheduling, inter-process communication, fault-tolerance mechanisms and synchronization. In addition, user implemented code that realizes the functionality for the application, like a controller function, is embedded into the generated code. The result of code generation is described in Section 2.5.

2.1 Modeling

The base for all modeling activities is the Eclipse Modeling Framework (EMF). With a suitable instantiator any meta-model, for instance UML, XML-Schema or EMF can be read. The model used to represent models in EMF is called Ecore. Ecore itself is an EMF model, and thus it is its own meta-model. Ecore consists of four classes to define a model: EClass, EAttribute, EReference and EDataType. The design of a meta-model is therefore similar to the design of a model. Thus, extensions of the meta-model can be realized very simple.

For modeling, Eclipse offers two possibilities: using the Reflective Editor, a tree-based modeling tool, or the graphical interface GMF.

Because the specification of a fault-tolerant system tangles different aspects, it is not useful to combine all of this information within one model. It must be rather possible to split up the models into distinct sub models that can reference each other. Referencing other models and meta-models is integrated within the Eclipse Modeling Framework. A detailed description of the used meta-models can be found in [1].

2.2 Validation

To find semantical errors, we have formulated several tests that check the semantical correctness of the specified models.

The specification of tests is supported in oAW by offering the validation language *CHECK*, an equivalent to the object constraint language (OCL) contained in UML. Tests in CHECK are specified as formulas in First-Order Logic. One example for a test is depicted in Figure 1: this test checks whether each declared interaction point, reflected in the concept port within our meta-model, is accessed by an actor object in read ac-

cess at least once. Other possibilities for tests are to rule out the unreachability of one application mode or to validate the correct usage of fault-tolerance mechanisms in relation to the used certification guideline.

2.3 Model-To-Model Transformation

After validating the different models, the models must be combined into one model. This model is used as starting point for the code generation. In addition, it is also necessary to calculate further information that helps to simplify the code generation. Such information is very often contained implicit within the model: e.g. if the model contains only a unidirectional reference, it might be useful to add also a reverse reference. This approach decreases on the one hand the error-proneness of the initial model, since directed references are much easier for the developer to maintain, while bidirectional references may get easily inconsistent. And on the other hand, we can benefit from bidirectional references when generating the code.
Model-to-Model (M2M)-Transformation is supported in oAW by offering the functional programming language *EXTEND*. In figure 1, the result of our M2M transformation can be seen in relation to the object port. We calculate, which objects access the port in read or write mode, on which electronic control units (ECU) the port must be available and assign a unique ID for every port object.

2.4 Code Generation

As described before, the code generation is based on templates. Templates represent the actual code generation ability and can be added very easily. Therefore, the extensibility of the code generator is guaranteed. Templates can be used to solve certain aspects of the run-time system, or to combine the results of different template to form a run-time system. Most templates are also platform dependent in the sense that they offer a solution only for a certain combination of hardware, operating system and programming language. Therefore, also the correct selection of adequate templates is necessary.
OpenArchitectureWare provides for these problems a special template language, call *XPand*. Xpand offers the statements *DEFINE* to declare a new code generation function and *EXPAND* to call other generation functions during the code generation. OpenArchitectureWare also allows polymorphism as one element to select adequate templates.
To specify the control flow of the code generation, the commands *FOR/FOREACH* and *IF/ELSE* can be used. The *FOREACH* statement is used to generate code for each object of a certain type that is declared within the model. Finally, the commands *FILE* and *ENDFILE* allow the management of the generated files. The code generation process is then rather simple: the adaptation of the templates to the model is performed using a technique similar to preprocessor macros. Text sequences between the different XPand commands are directly copied to the generated files and variables allow the access to objects and their attributes.
Figure 1 shows one example for a template that is used to generate code for the declared ports.

2.5 Code Generation Result

Usually, the generated files contain source code for an arbitrary programming language. But since oAW is not restricted to one specific output language, it is also possible to generate documents in natural language. This can be useful if documents for certification issues or user-manuals are required.
In the current version of our tool, we are generating an executable run-time system. This system includes code for the timely-correct execution of the application, for process management and scheduling, as well as communication (interprocess, interprocessor) functionality. In addition, the selected fault-tolerance mechanisms are realized by the run-time system. The actual code realizing the application functionality, like control functions, are not covered by the tool, due to the existence of several other tools like Matlab/Simulink or SCADE. The application code is embedded into the run-time system during the code generation process.

3 Application Examples

To point out the advantages of our approach, we implemented two demo applications with our development tool: an elevator control and a time-critical control application.

In our lab courses, we are using elevators build up with Fischer-Technik components and self-made circuit-boards containing a microcontroller for each floor, the cabin and the cabin motor. All microcontrollers are connected via CAN-Bus. Each circuit-board acts as an actuator and/or sensor and controls the door, lights, safety-light barrier and LED-Display of each floor, the keys for appointing the movement direction and the cabin's indoor control panel. Furthermore, there are sensors mounted inside the elevator to report the actual position of the cabin.

Two standard PCs are connected to that CAN network and perform the control task as a hot-standby system. The units perform liveliness tests to mutually supervise each other. In case of a missing live sign of the main unit, the backup unit will take over control. In addition, an error message is send to a third unit performing the monitoring of the system. If instead the backup unit fails, only the error message is send to the monitoring unit.

At begin of the execution, each unit will perform tests, whether the other unit is already performing the control task. If this is true, the unit will send an integration request, will eventually receive the information necessary for state synchronization and will integrate into the system. Note that the fault-tolerance mechanisms guarantee that exactly one control unit will send the control messages to the microcontrollers.

If the other unit is not already performing the control task, the unit will try to synchronize with the other units and start the control function.

In the second application, we implemented a control function on a Triple-Modular Redundancy system. We used for this application, three control units with Switch Ethernet as communication medium. The task was to balance a rod by controlling switched solenoids. For this task, control rates of 1 millisecond were necessary. To achieve consensus on the error states of the units, we used interval voting. Here, the user had to specify the allowed deviation of correct values. In addition, mechanisms to exclude units with erroneous results and to integrate repaired units into the running system were generated.

The usage of our tool was very successful in both applications. The code that had to be implemented by the application developer was limited to 50 (control function) to 500 (elevator control) lines of code. The vast difference between both applications can be explained by the necessary bit manipulations of the elevator control due to the application protocol, as well as the greater number on input and output functions.

4 Conclusion

In this paper, we have presented an approach to generate system-level code using a model-based approach. The main contribution was to point out the possibilities that are offered by existing template based code generators, in our example the openArchitectureWare MDA/MDD suite.

OpenArchitectureWare offers all the possibilities needed for the generation of such code: multiple models also using different meta-models are supported, tests can be specified easily by the use of a language based on First-Order Logic, model-to-model transformation is supported to combine the different models and a template language can be used to implement components that can be adapted to specific models.

The greatest benefits of this approach are the extensibility of the underlying meta-models and the code generation ability, the use of templates for various system aspects that can be extended later on and the opportunity for intuitive modeling of complex embedded systems. The overall approach has been already tested in some lab applications, like a pending pendulum and an elevator application that we have described shortly in this paper.

The next step is to apply our approach within a real industrial project to point out the feasibility of the approach. In addition, we want to use formal methods to verify the correctness of the generated code. Here, a first step can be to prove the adequacy of the selected fault-tolerance mechanisms in relation to the expected faults.

References

[1] C. Buckl, M. Regensburger, A. Knoll, and G. Schrott. Models for automatic generation of safety-critical real-time systems. In *Second International Conference on Availability, Reliability and Security (ARES 2007)*, pages 580–587. IEEE Computer Society, Apr 2007.

[2] A. Deursen, P. Klint, and J. M. Visser. Domain-specific languages. Technical report, CWI (Centre for Mathematics and Computer Science), 2000.