Using MARTE in Code-centric Real-time Projects Providing Evolution Support

Peter Ulbrich*, Christoph Elsner†, Martin Hoffmann*, Reiner Schmid†, Wolfgang Schröder-Preikschat*
* *Friedrich-Alexander University Erlangen-Nuremberg, Erlangen, Germany*
{ulbrich,wosch}@cs.fau.de, martin.hoffmann@e-technik.stud.uni-erlangen.de
† *Siemens Corporate Technology & Research, Erlangen/Munich, Germany*
{christoph.elsner.ext,reiner.schmid}@siemens.com

Abstract

The MARTE UML profile targets specification and analysis of real-time systems. It has been used both, in model-driven development (MDD) approaches incorporating code-generation and in approaches that manually reengineered existing systems. The first option is not possible in many cases, as certain regulations, the preferences of the developers, or substantial legacy code may impede it. The second option, in contrast, constantly involves the risk that code and models drift apart during system evolution, necessitating expensive, manual code-model resynchronisation and impeding the day-to-day analysis of the evolving system.

In this paper we present an approach for modelling existing real-time systems with MARTE. It provides semi-automated support for synchronous evolution of code base and model without the need to switch to a code-generation–based MDD approach. We illustrate our approach by describing several evolution scenarios and discuss the most critical issues.

1. Introduction

The MARTE UML [7] profile is a de facto standard that provides a specification and analysis model format for safety-critical, real-time, and embedded systems.

One way to use MARTE is to consider model-driven development (MDD) right from the start applying a top-down approach [1], [5], which includes automatic model-to-code transformation. This way, functional as well as non-functional properties of the system are covered within models and analyses can be performed on a day-to-day basis, minimising the impact of bugs and design errors.

However real-world real-time systems often have to consider legacy code, domain-specific models (e.g., Matlab Simulink), or certification standards [4]. Equally, the skills and preferences of the involved system experts may impede the use of MDD. Driving top-down MDD approaches from the start, or even redesigning code-centric development approaches to MDD, for example by using tools such as Fujaba [1] that allow for keeping Java code and UML models in synchronization semi-automatically, are therefore often not possible.

One possibility to cope with this is to reengineer crucial parts of existing systems manually with UML and MARTE without switching to MDD [2]. This means extracting the system structure and non-functional properties, such as the worst-case execution times (WCETs) or resource usage, from the code and from specifications and thus making them explicit. However, the code base is under constant change. Consequently, the model either goes out of sync with the code base or has to be adapted. We are not aware of any approach using UML MARTE for reengineering that actually broaches this issue. We assume that, up to now, existing reengineering approaches bring model and code to sync at specific moments in time only (e.g., prior to a new release). As model and code then already may have drifted apart considerably, those approaches then cannot provide comprehensive support for resynchronisation.

Keeping models and code in sync has remarkable benefits: the current system can be analysed on a day-to-day basis, and programming and design errors may be discovered early. If code and models are not too much out of sync, it is possible to give semi-automated support to the usually manual and tedious resynchronisation tasks, for example by extracting information from the code into the model or by reproducing simple refactorings (e.g., method signature changes). Other changes may require human intervention; it is however possible to detect *that* this is the case.

In this paper we will follow exactly this approach: An existing real-time project shall constantly benefit from the options that MARTE offers, in consideration of an evolving code base and without having to use model-driven development.

In this paper, we present our approach, illustrate it with several some evolution scenarios, discuss it, and report on our ongoing implementation.

2. Approach

In this section we present our approach for using MARTE in non-model-driven real-time settings, thereby supporting

This work is supported by the Embedded Systems Institute (ESI) – http://www.esi.uni-erlangen.de

[1] The Fujaba Project - http://www.fujaba.de/

the synchronous evolution of models and code base. Unlike MDD, it only requires minimal changes to existing, code-centric development practices and aims at giving semi-automated support for synchronising structural, functional, and non-functional properties of the system.

We see the source code base as our central information repository and therefore drive a bottom-up approach (Figure 1). First (Step 1) we reengineer the UML model and enrich it with MARTE data provided by automated analysis tools working on the code base or the binary data, respectively. In Step 2, we model the MARTE data, that cannot be retrieved automatically, manually in a separate model and merge this with the enriched UML model. Finally, in Step 3, we are able to run analyses such as a schedulability analysis on the resulting, integrated MARTE model. In the following, we describe the three necessary steps in detail:

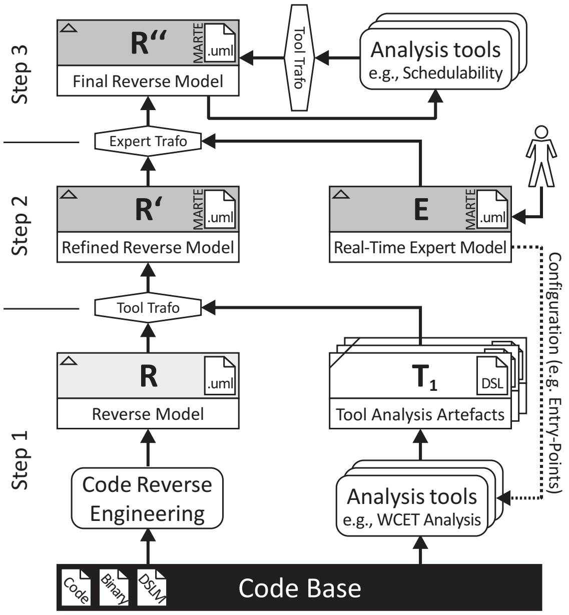

Figure 1. Reverse Approach Overview

Automatic model generation (Step 1): The initial UML model R (*Reverse Model*) is generated using a code reverse engineering tool, which exists for both C and C++ as the most common programming languages for real-time systems. For creating the *Tool Analysis Artefacts* ($T^* = \{T_1,..,T_n\}$) we use analysis tools specially targeting the real-time properties of the source code, the executables, and the domain-specific models (e.g., Simulink) that are flanking the code base. Various non-functional system properties can be extracted: WCET, resource usage, the assignment of program code to tasks, or the current schedule. A model transformation (*Tool Trafo*) reads the information from the artefacts in domain-specific notation and integrates it into the model R, applying appropriate MARTE stereotypes to it thereby creating the model R' (*Refined Reverse Model*).

Manual modelling (Step 2): Many real-time properties are only implicitly available and cannot be extracted from the source code automatically. Deadlines or minimal inter-arrival times are, for example, determined by the physical environment in which the system is embedded, whereas the maximal jitter for a certain task might be dictated by a control engineer. Various kinds of analyses, such as schedulability analysis, however, require such information.

For making the implicit information explicit, we introduce the model E (*Real-Time Expert Model*), holding those properties. Furthermore, it can be used for introducing logical (e.g., as in [8] for timing issues) and architectural views on the system for model-to-code transformations in future. Finally, its information is fed back to the previous step to configure the analysis tools (e.g., declaring entry points for WCET analysis).

The two models R' and E have to be combined by the model transformation *Expert Trafo*, which is a crucial part of the approach. It relies on a reference mechanism that relates elements from E to R' and a constraint checking mechanism that identifies inconsistencies in the case that R' evolves. Only if all constraints are fulfilled, the *Expert Trafo* actually may merge R' and E. A detailed discussion of the projected implementation will follow in Section 5.

Analysis (Step 3): The combined model R'' (*Final Reverse Model*) serves as input for further analysis tools. Another model transformation integrates the result back into R''. At this point the MARTE standard can play to its strength. More and more tool vendors are presumed to adopt it and analysis tools may be successively added. Dependent on the level of detail and the type of the analysis desired, more information may be required in R''. For this purpose, it is possible to refine the approach in an iterative way by adding more analysis tools during Step 1 or provide more implicit information in Step 2.

Summarising, our approach combines a non-model driven development approach with a synchronised code base and MARTE models, which is necessary for running regular real-time analyses to detect programming errors and design flaws early. It provides support for automated real-time property extraction and a merging mechanism to include implicit expert knowledge.

3. Target System

The evaluation platform of our MARTE research is the *I4Copter* [9] quadrotor helicopter. The *I4Copter* has been designed and developed to resemble embedded real-time systems arising in real-world industrial scenarios. Therefore it uses an Infineon TriCore microcontroller commonly used in automotive ECUs and a custom made sensor periphery board featuring a wide range of sensors (12 in total) with a variety of interfaces (analogue, digital, SPI and RS232).

Figure 2. The *I4Copter* Quadrotor Helicopter

The system itself has been developed in a classical (non MDD) way in our Real-Time System Lab. In total, the application comprises ∼7000 LOC, is written in C++ and features an object-oriented, component-style architecture with high coherency and minimal coupling among the components. There are periodical (e.g., flight control) and sporadic (e.g., wireless remote control) real-time tasks having firm as well as hard deadlines.

The tasks, executed by the real-time operating system *PxROS-HR*[2], are scheduled statically and ensure isolation using hardware memory protection.

4. Evolution scenarios

Basic Example - Schedulability Check: To perform a schedulability check of the system, the analysis tool requires diverse information about the involved tasks. The basic UML model of the system's entities, such as the existing tasks and entry points, can be reverse engineering automatically from the code base (Section 2, Step 1). By convention, all tasks in the system have to inherit from a *Task* class declaring an *entry()* method. A model transformation, thus, can immediately identify the tasks out of the reverse engineered model and apply the stereotype `<<swSchedulableResource>>` to the corresponding classes.

Other information needed for a schedulability check, however, cannot be reverse engineered from the code base. The corresponding properties have to be modelled manually in the model E (Section 2, Step 2). The minimal and maximal period of a task are essential properties. Given a task gathers data from a sensor. One the one hand, the sensor has a maximal sampling rate (noted in datasheet); this is an indication for the minimal period. The maximal period, on the other hand, depends in the case of the I4Copter on the maximal possible period of the flight control procedure that still provides adequate flight behaviour. Both values cannot be deduced automatically and, therefore, a real-time expert has to contribute them manually. Additionally, annotating WCETs is mandatory for schedulability analysis. It can be determined from the binary data by an appropriate analysis tool in Step 1.

After merging all information from R' and E, the final reverse model R'' contains all necessary information for performing a schedulability check (Section 2, Step 3).

Evolution Example 1 - Small Change: Consider the signal processing developer replaces the source code of a filter algorithm with one having a much higher computation time. At the push of a button, the toolchain can evaluate whether the system can meet all timing constraints.

Evolution Example 2 - Considerable Change: A more profound modification of the system than just altering some task's code is the replacement of a hardware sensor. Consider a sensor previously attached to the system by an analogue interface is replaced by a sensor communicating on a shared medium (e.g., SPI or CAN). This has a major influence on the system's schedule, since the task using this sensor is now closely linked to all other entities using the shared medium.

First of all, a representation of the shared resource has to be introduced into the reverse model. This can be achieved automatically, given all classes using a shared resource inherit from a class *SharedResource*.

Second, a new analysis tool is necessary to detect access sequences to the shared resource. This might also involve adding a new type of model, e.g., for specifying system behaviour. If the schedulability check discovers that a non-preemptive schedule cannot meet all timing constraints any more, the scheduling paradigm might need to be changed to a preemptive priority-ceiling protocol for shared resources. Finally, if the currently used schedulability tool is not able to handle shared resources, it can be replaced by a different tool supporting MARTE as import format, without the need to change models or write new converters.

5. Implementation

We are currently implementing the toolchain for our approach and started creating the expert model for the quadcopter system. We used *BoUML*[3] for reverse engineering the C++ sources and *AbsInts aiT*[4] for the WCET analysis. The model-driven toolchain is based on Eclipse and its modelling facilities[5], in particular, openArchitectureWare[6].

Whereas the *Tool Trafo*, which simply annotates the basic UML model with MARTE stereotypes and tagged values, is a rather simple model transformation, the *Expert Trafo*, which has to cope with evolution of the code base, is more challenging. It relies on a *reference mechanism* and a *constraint checking mechanism*.

Reference Mechanism. Maintaining stable references from model elements of E to the evolving model elements R' is not trivial. The XMI reference IDs created by standard tools are unstable and tool dependent; reengineering R' from

[2] HighTec EDV Systeme GmbH - http://www.hightec-rt.com/

[3] BoUML Toolbox - http://bouml.free.fr/

[4] AbsInt Angenwandte Informatk GmbH - http://www.absint.de/ait/

[5] Eclipse Modelling Framework - http://www.eclipse.org/emf/

[6] OpenArchitectureWare - http://www.openarchitectureware.org/

a slightly different code base may result in a complete invalidation of all references in E. Shifting references to a separate mapping model, which identifies the elements of each model to map using their fully qualified names, is a more stable option. However, we decided that for each model element of R' that needs to be annotated, there has to be one element in E having exactly the same package, type, and name. Doing so, it is possible to use a simple merge algorithm to combine E and R'.

Constraint Checking Mechanism. Coping with an evolving code base has not only real-time–related challenges (as discussed in Section 4) but also rather technical ones. Adding, removing, and renaming of classes, methods, or members bring the models out of sync, as well as changing code from which a behavioural model is reengineered. Detecting these cases is however relatively easy by defining OCL-like constraints that enforce plausibility checks. For example, one constraint will be that each method defined and annotated in E needs a correspondence in R'. The checks will be run each time before each execution of the *Expert Trafo*, and thus alleviate the risk of merging inconsistent models. Further tooling might support the export modeler with (semi-)automatic functions for adapting E to a changed R'. This is, however, out of the scope of this paper.

Expert Trafo As we require the expert model E to resemble the structure of R' for those elements needing annotations, we can use a simple merge semantic for combining the models. Fortunately, UML2 already provides the *Package Merge* operator in its Superstructure definition [6]. The Eclipse Modeling Framework comprises an implementation of the operator's algorithm. Although *Package Merge* has been designed to merge the UML meta model specification and it is not free of semantic inconsistencies [3], for our current modelling purposes it is absolutely sufficient, as we only need the basic recursive merging, which incorporates merging of stereotypes.

6. Discussion

In the following, we will discuss the quality of the reengineered model R from source code and the feasibility of keeping the models E and R' synchronous.

Reverse engineering UML models out of existing code is prone to errors in the general case. Nevertheless, in safety-critical real-time applications, strict processes and programming rules (e.g., regarding pointers) have to be applied to ensure high quality of the code. Further on, such applications are of a completely static nature regarding their composition and interactions. This way, we can exclude a lot of pitfalls (e.g., dynamic message passing) that are commonly difficult to extract during UML reengineering.

We address synchronisation of the models E and R' with a constraint checking mechanism and we have presented how we intend to approach different kinds of evolution scenarios. Substantial changes, such as the introduction of shared resources described in Section 4, Example 2 could compromise the toolchain, as necessary analysis tools and checks are not present. However, as our reengineering approach has to follow the same stringent development processes as the rest of the system; those cases can be detected and handled by introducing the appropriate tools.

Despite this support, our approach still requires substantial manual involvement of the real-time developers, thus, the ultimate proof of feasibility will be acceptance of the developers.

7. Conclusion and Outlook

In this paper we presented an approach to for applying MARTE to an existing, code-centric real-time system considering the evolution of the code base and without switching to an MDD approach. We illustrated our approach by describing several evolution scenarios and discussed the most critical issues.

Once implemented, the approach will allow us to detect programming errors and design flaws early. Even more, we expect that it will be highly useful considering the future plans for the *I4Copter* project. There already exist three quadcopters, each comprising different hardware. Further upgrades will include cameras and GPS hardware. This varying hardware will be accompanied by differing application software, as, depending on the operation area (e.g., security, rescue, hobby) of a quadcopter, different functionality will be necessary. With the growing number of hardware and software variants, the effort for manually analysing and planning the real-time properties of different variants will become more and more infeasible. Still, our approach will have to be extended considerably to cope with such a situation and there are still various challenges regarding the combination of real-time modelling, analysis, and variability on the way. However, we are confident that by committing us to automating the analysis process we are making an important step into the right direction.

References

[1] D. Aulagnier, A. Koudri, S. Lecomte, P. Soulard, J. Champeau, J. Vidal, G. Perrouin, and P. Leray. SoC/SoPC development using MDD and MARTE profile. In *Model Driven Engineering for Distributed Real-time Embedded Systems*. Hermes, 2009.

[2] S. Demathieu, F. Thomas, C. André, S. Gérard, and F. Terrier. First experiments using the UML profile for MARTE. In *11th IEEE Int. Symp. on OO Real-Time Distributed Computing (ISORC '08)*, pages 50–57, Washington, DC, USA, 2008. IEEE.

[3] J. Dingel, Z. Diskin, and A. Zito. Understanding and improving UML package merge. *Software and Systems Modeling*, 7(4):443–467, October 2008.

[4] M. U. Khan, K. Geihs, F. Gutbrodt, P. Gohner, and R. Trauter. Model-driven development of real-time systems with uml 2.0 and c. pages 33–42, Washington, DC, USA, 2006. IEEE.

[5] C. Mraidha, Y. Tanguy, C. Jouvray, F. Terrier, and S. Gérard. An execution framework for MARTE-based models. In *13th Int. Conf. on Eng. of Complex Comp. Sys. (ICECCS '08)*, pages 222–227, Washington, DC, USA, 2008. IEEE.

[6] Object Management Group OMG. Unified modeling language (UML) 2.1.2 superstructure specification. formal/2007-11-02, November 2007.

[7] Object Management Group OMG. UML profile for MARTE, beta 2. ptc/2008-06-08, June 2008.

[8] M. Peraldi-Frati and Y. Sorel. From high-level modelling of time in MARTE to real-time scheduling analysis. *1st Int. W'shop on Model Based Architecting and Construction of Embedded Systems (ACES 08)*, pages 129–143, 2008.

[9] P. Ulbrich. The I4Copter project — Research platform for embedded and safety-critical system software. http://www4.informatik.uni-erlangen.de/Research/I4Copter/, visited 2010-02-22.