Developping Real Time Embedded Applications Independently of an Execution Platform

Julien DeAntoni
INRIA Sophia Antipolis - AOSTE project
2004 route des Lucioles
BP93, F-06902 Sophia Antipolis Cedex, France
julien.deantoni@sophia.inria.fr

1. introduction

To address the growing complexity of Real-Time and Embedded (RTE) systems, a know strategy is to combine software engineering principles and formal techniques. The software engineering principles should allow reuse of software parts whereas formal techniques should ensure the correctness of the system. During last years, an emerging idea is to use, among the various software engineering approaches, Model Driven Engineering (MDE).

In order to guarantee the system correctness, the models associated with the existing tools [12, 9, 10] and paradigms take into account the properties of: the application, the execution platform and the environment. After simulation and/or formal analysis, they are able to fine-tune the system parameters in order to reach the desired characteristics. It results in a rigid system that makes the reuse of system parts difficult (and sometimes impossible) [2].

On the other hand, CBSE (Component Based Software engineering) proposes to decompose a system into various, possibly independently developed, components. The components have well defined interfaces, which are connected to compose the system. The composition correctness can be verified by establishing contracts during the composition [4]. Few CBSE approaches are dedicated to RTE systems. Moreover, these approaches either do not use QoS contracts on their interface or only consider simple and directly composable properties [6] such as the memory usage. Consequently, it is still difficult, in a RTE context, to reuse some software parts safely, e.g. by guaranteeing the use of a RTE application on a specific execution platform.

This paper briefly explores the notion of abstract platform for RTE applications and illustrates it through SAIA (a CBSE and MDE approach for the specification of control system independently of the environment communication platform).

2. Ongoing Research Approach

The general idea consists in using MDE to abstract real time and embedded systems. In recent approach such as MARTE [11], the abstraction separates the software part and the execution platform part.

Software and execution platform models conform to their own metamodel but are issued from a same set of concepts, mainly for the representation of the component model (see top center of the figure 1). In these metamodels, the properties which are important for analysis must be specified. This way, it is possible to realize one or more intra model analysis in order to evaluate software or execution platform independently (left and right part of figure 1). One can notice that in order to realize the analysis, a model transformation is needed between the model to analyze and the formalism of the analysis language. A requirement on these transformations is the respect of the operational semantics of the input model. In additional to these analysis that are realized on a single model, it must be possible to realize inter model analysis, i.e. in our case, analysis that take as input a software model and an execution platform model (see figure 1). These analysis may ensure, for example, that the execution platform is compatible with the software and conversely. Once again, a transformation must be realized between each model and the analysis language formalism. When these steps result in correct models, each of the models must be refined in order to incrementally produce the final realization of the system. For each of the refinement steps, the intra and inter model analysis must be realized in order to verify the correctness of the refinement ([8]).

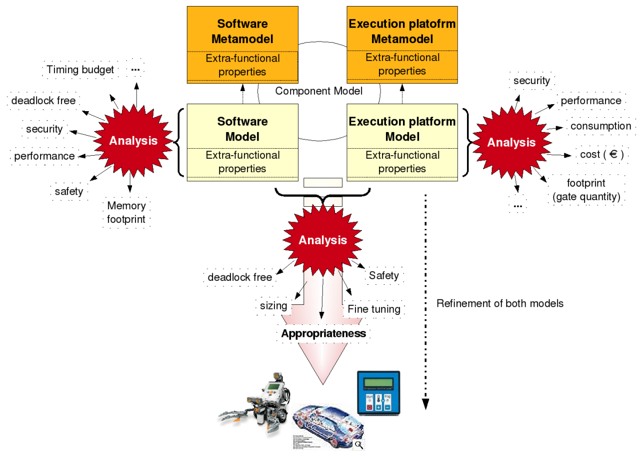

Figure 1. Global approach for real time embedded system development

Even if this approach is still under study, some of the current formalisms enable this kind of development [11, 13, 3]. Unfortunately, these approaches, where the software parts are refined accordingly to the execution platform and reciprocally, lead to a software model and an execution model that have been fine tune and have became specific to each other. Once again, reusing the software parts in a different execution context is difficult. Moreover, since these approaches do not specify clearly what is non-functionally expected by one model from an other; inter-model analysis must be realized for each refinement step. These inter model analysis can quickly lead to combinatorial explosion if the models are two detailled (details appear with refinement). The next section highlights how the use of an abstract platform addresses these issues.

3. Toward the use of an abstract platform

To simplify the reuse of software from an execution platform to another one, we argue that the definition of an abstract (sometimes called virtual) platform is crucial. An abstract platform defines a set of acceptable platforms from an software point of view. An abstract platform defines characteristics that may be allocated onto a set of concrete platforms that are considered as potential targets in a development project [1]. In the context of RTE systems, the abstract platform must also contain the extra-functional properties for which the software is correct. In other words, it must specify the minimal (and maximal) extra-functional properties that must be guaranteed by a concrete execution platform to ensure the system correctness. On the other hand, the actual extra-functional properties of a concrete platform, that results from intra-model analysis, must be given. Then, a notion of extra-functional contract must be introduced to verify if the extra-functional properties specified by an abstract platform are satisfied or not by a specific concrete platform. In some extent, the abstract platform and the required extra-functional properties can be seen as a way to define the design space of the concrete execution platform. Rather than ending with a specific platform like in classical design space exploration, we define a set (hopefully not empty) of concrete platforms suitable to our application. The abstract platform is then a central point where functional and extra-functional information needed for the software model and the execution platform model to work each other are captured. To define an abstract platform for RTE applications, we have to provide: (1) a way to specify extra-functional constraints and (2) a way to verify if a contract can be established or not, which depends on the nature of the extra-functional properties.

This approach has been partially implemented in SAIA [7], focusing on the part of the abstract platform relating to the communication between the software and the environment.

4. SAIA

SAIA defines an abstract platform in terms of *Inputs* and *Outputs* (I/O), which are a specification of the communication between the application and the environment. I/O represent physical values or actions in the environment but do not give information on how their are produced or realized. For instance, an *Input* of an exploration robot can be its *Speed*. On the other hand, a concrete platform is a set of *Sensors* and *Actuators* (S/A) that specifies a specific way to access to physical values or to realize actions in the environment. For instance, a *Sensor* named *WheelRotation* can generate an event at each wheel rotation.

Due to the difference between information in the abstract platform and information in the concrete platform, some adaptations are mandatory. For instance, the *Speed Input* can be computed from the *WheelRotation* information but an interpretation must be done. These adaptations are realized by a complex connector, i.e. a connector augmented with a behavior.

Since RTE system correctness depends on the timing properties of the data flows from and to the environment, the properties for which the system is correct are defined in the abstract platform. On the other hand, the actual properties of a concrete platform are also described, after analysis. Before any attempt to establish a contract between the abstract and the concrete platform, the temporal impact of the complex connector must be evaluated. This evaluation results in a modification of the previously defined properties that takes into account the complex connector behavior. Then, an attempt to establish a contract is made in order to check if the specific concrete platform satisfies the abstract platform, and so, ensure the system correctness. It is important to notice that this latter check (the contract establishment) is only based on the previously described properties and no longer on any behavior.

To realize each of the previously presented points, SAIA encapsulates into models: (1) a formal language based on timed automata for the behavior description[1]; (2) a formal specification of the extra-functional properties based on real-time logic formula and (3) a formal analysis for the specification of the contract specification. The case studies addressed by SAIA have shown the ability to reuse, without any change, the same RTE application on different S/A platform as well as the ability to predict correctness of the system thanks to contract establishment.

5. Conclusion

The use of an abstract platform conjointly with MDE techniques seems promising to enable the development of RTE application independently of a specific execution platform. SAIA is a first attempt to specify and illustrate the different mandatory steps involved in the use of an abstract platform for RTE systems. However, SAIA covers only the environment communication aspects of a RTE application. The work must be done for execution as well as network aspects. We currently investigate how precise time description formalisms like MARTE [11] can give the basis for the construction of such aspects.

References

[1] J. Almeida, R. Dijkman, M. van Sinderen, and L. Pires. On the notion of abstract platform in MDA development. *Enterprise*

[1]this language is a subset of the IF language, allowing the use of associated tools [5] by model transformation

Distributed Object Computing Conference, 2004. EDOC 2004. Proceedings. Eighth IEEE International, pages 253–263, 2004.

[2] ARTIST. *Adaptive Real-Time Systems for Quality of Service Management*, pages part 1 – chapter 3. 2003.

[3] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli. Metropolis: An Integrated Electronic System Design Environment. *COMPUTER*, pages 45–52, 2003.

[4] A. Beugnard, J.-M. Jézéquel, N. Plouzeau, and D. Watkins. Making component contract aware. *IEEE computer, 32(7)*, pages 38–45, 1999.

[5] M. Bozga, J. Fernandez, L. Ghirvu, S. Graf, J. Krimm, and L. Mounier. IF: A Validation Environment for Timed Asynchronous Systems. *Proceedings of the 12th International Conference on Computer Aided Verification*, pages 543–547, 2000.

[6] I. Crnkovic, M. Larsson, and O. Preiss. Concerning Predictability in Dependable Component-Based Systems: Classification of Quality Attributes. *lecture notes in computer science*, 3549:257, 2005.

[7] J. DeAntoni. *SAIA : un style architectural pour assurer l'indépendance vis-à-vis d'entrées / sorties soumises des contraintes temporelles*. PhD thesis, Institut National des Sciences Appliquées de Lyon, Octobre 2007.

[8] D. Densmore. Formal refinement verification in metropolis. Technical Report UCB/ERL M04/10, EECS Department, University of California, Berkeley, 2004.

[9] dSPACE. Powerful tools for controller development. http://www.dspaceinc.com/ww/en/inc/home/products.cfm?nv=n2, 2007.

[10] esterel technology. Scade suite (tm). http://www.esterel-technologies.com/products/scade-suite/, 2007.

[11] F. Mallet and R. de Simone. Marte: A profile for rt/e systems modeling, analysis (and simulation?). In *SIMUTools'08*, Marseille, France, March 2008.

[12] Mathworks Inc. MATLAB. *http://www.mathworks.com/products/matlab/*, 2005.

[13] J. Stankovic. VEST-A Toolset for Constructing and Analyzing Component Based Embedded Systems. *Proceedings of the First International Workshop on Embedded Software*, pages 390–402, 2001.