Definition of the Virtual Prototyping Framework FoToVP Deliverable #1

Florence Maraninchi, VERIMAG
Jean-Pierre Talpin, IRISA

Decembre 15th, 2007

1 Introduction

We first recall the objectives of the project.

In the context of past or current projects involving industrial partners from various application domains, the participants of FoToVP have observed several approaches for the design of complex and/or critical embedded systems, based on the notion of virtual prototyping. This allowed us to identify clearly where there is a need for formal tools. We started studying the benefits of formal methods and tools in the other projects, with the constraints of particular application domains, and with practical objectives in mind. Some recurring problems appeared, that need to be investigated further, independently of these application domains, and with less constraining short-term practical objectives. In this project called FoToVP, standing for "Formal Tools for Virtual Prototyping of Embedded Systems", we would like to study these recurring problems, in order to develop more fundamental and generic results. The motivations are clearly related to industrial applications, and the applicability of the project results will be evaluated with respect to these industrial practises and applications. The three application domains we have been studying in other projects, with industrial partners, are:

- Systems-on-a-chip, with STMicroelectronics
- Sensor networks, with FranceTelecom R&D
- Embedded control systems, with EADS, Airbus, RATP,

In all these domains, we observed that the design of complex systems relies on the definition of a virtual prototype, in the form of some executable high-level model of the system. This is made complex because such a model has to cover all elements of the embedded system, at various levels of abstraction: the hardware pieces, the software (both application software and operating system or middleware levels), the execution platform, and the physical environment. First, an executable model may be used extensively for simulations, to observe functional and non-functional properties of the system under construction. To our opinion, the main points to be studied and integrated in a generic prototyping framework are:

- Accurate modeling of physical environments and execution platforms
- Extraction of rich executable formal models from non-formally defined proto- typing languages like SystemC, Java, etc.
- Automatic abstraction of these rich models, according to various criteria
- Integration of external black-box code

- Modeling of non-functional properties (non-functional time, energy, ...)
- A general formal framework for functional and non-functional modular ab- stractions

2 Definition of the Virtual Prototyping Framework (VPF) in FoToVP

The first discussions in the project have led to the main choices for the Virtual Prototyping Framework (VPF) we will study. We have also refined the objectives of the project.

2.1 Needs and Global Picture

A VPF includes all elements that allow to simulate the complete system: the hardware, the software (both application and infrastructure software), a model of the physical environnement of the system, a model of the communication media.

There is a clear distinction to be made between: 1) *models* in the sense of *physical models*, i.e., mathematical or computer models that represent some physical reality; 2) *models* for hardware and software, in the sense of so-called "model-driven" development approaches, i.e., abstractions of some computer system element. For the latter, it is possible to include in a VPF the exact definition of the computer system element (a VHDL description of the hardware, the exact code of a protocol, etc.), but the difficulty comes from the need to use more abstract models. For the former, a model will always be some approximation of the reality, and the difficulty is to design a faithful model.

We review all the elements of a VPF below, and comment on the type of model we need, and on the methods to be used to obtain these models.

2.2 Modeling the Physical Environment

For modeling the physical environment, we will use executable models described in a synchronous language (Lutin, Lustre or Signal). Experiments already exist, and since the synchronous languages are formally defined, it will be easier to use in a formally defined context.

The choice is mainly between: 1) executable models; 2) equational models. When the physical phenomenon to be modeled is well known, and has been described by equations with perturbations, solution 2 is the right one. But the physical environment to be included in our VPs are seldom well defined, and it is often easier to describe them directly in some operional way.

For instance, the physical environment of a sensor network is the source of all the stimuli on the sensors. If we consider a pollution cloud, the stimuli are strongly correlated both spatially and temporally (if the cloud is somewhere at a given instant, it is likely to be close tto this place a small amount of time after that). Describing these correlations is easy if we "program" the cloud, as a non-deterministic behavior.

This has been described in [5]. The complete integration of such environment models in functional models of sensor networks will be part of the PhD of L. Samper at Verimag, available next semester.

2.3 Modeling the Execution Platform

The notion of *execution platform* for embedded software may cover various situations. For instance, it may be the set of hardware components plus the operating systems, in a system-on-chip. It may also be the hardware of the nodes, plus the radio channel, in a sensor network. In a distributed system, it includes the computing units, plus the communication elements (buses, ...).

Consequently, the model of the execution platform that has to be included in a global virtual prototype of an embedded system may need: 1) the modelling of some physical phenomenon (e.g., the radio channel); 2) the modelling of dedicated hardware; 3) the modelling (or the inclusion in the model) of the infrastructure software.

For case 1), see the previous section on physical models. For cases 2) and 3), the problem is either to build models from scratch, or to extract models from existing languages which are not necessarily well-defined. See next subsection.

As demonstrated by the related work [2, 3], the environment of the system and the execution platform can be specified using AADL [1] and then interpreted in a synchronous model of computation and hence incorporated and understood in the present virtual prototyping framework.

The use of similar techniques is under study at VERIMAG, for the fine description of a the hardware architecture of the ATV (Automatic Transfer Vehicle). This will be available as a master student report at the end of june 08.

2.4 Model Extraction for Hardware and Software Descriptions

Model extraction is the operation that produces a formal model (e.g., that can be exploited by some automatic verification tool like a model-checker), from the description of a computer system element, in some programming or hardware-description-language (HDL).

In FoToVP we will concentrate on the extraction of models from languages of the C family (C, C++, Java, SystemC). For the software part it is representative of the languages used in the embedded system domain. For the hardware part, it is representative of the languages used for the description of hardware at a higher level of abstraction than the Register-Transfer-Level which is the entry point of synthesis tools.

For the hardware part, one important need is to be able to extract a model of the *architecture* of the system.

3 First Unifying Task

This first view on the needs for FoToVP has led to the definition of the first unifying task: the definition of a formal model that could be used as a central format for all the models described above.

Comparing the experiments made previously by VERIMAG and Espresso, we decided to study the definition of a formalism based on automata and products of automata, with the following characteristics:

- The basic automata should be usable for the description of software at any level of details. Hence they should be interpreted automata, with no limitation on the expres-

sive power. However, an interpreted automaton allows a clear distinction between a finite control structure, and a set of variables of any type on which all operations are permitted. This form of description is adequate for the use of abstract interpretation tools.
- These basic automata should be able to describe the notion of atomicity needed in any parallel language. We suggest two kinds of states, as already explored with MicMac automata in the PhD of Jérôme Cornet at VERIMAG (available next semester). The idea is to distinguish between *micro-states* that represent the internal activity of a process, and *macro-states*, that represent the points at which a process may yield to the scheduler (or let the other activities change the global state). This kind of automaton with two kinds of states may be used to describe the semantics of the non-preemptive scheduler of SystemC. It can also be used to describe the semantics of parallel threads in Java: the `synchronized` keyword identifies pieces of code that have to be translated by micro-states.
- The products should be able to describe both the pure synchronous composition, and the various kinds of asynchronous compositions.

The first technical work will be on the redefinition of the HPIOM formalism (as defined in the PhD thesis of Matthieu Moy [4]), and the integration with what can be done by using the SSA form of the GCC compiler to obtain models extracted from software. In HPIOM, the basic steps of a C++ program first give successive transitions in the automaton; then some transitions are merged if they represent code that can be executed in parallel.

The translation of the SSA form into Signal is done the other way round. It exploits the structural properties of the control-flow in an SSA program: since all variables on a path of the control-flow are guaranteed to be assigned at most once, each path can be translated by the composition of data-flow equations. The translation of the SSA form hence consists of interpreting each instruction of the SSA program by an equation of equivalent meaning. This results in the exposition of maximal parallelism and gives opportunities for verification and optimization.

Although the two methods are different, and may be compared on performance criteria (time, size of the automaton produced, ...), they rely on the same idea: an automaton representing a sequential program should expose maximal parallelism.

4 Conclusion

The various aspects of the project have been clarified, and several subtasks have been started (modeling the architecture, moldeing the physical environment, studying special kinds of automata for the representation of various synchronous and asynchronous frameworks).

A special task has been started, on the comparison between the HPIOM approach (Verimag) and the approach based on SSA (Espresso), for the prodution of automata that encode pieces of programs written in languages like C, C++ or Java.

References

[1] Peter Feiler. SAE AADL: An industry standard for embedded systems engineering.

[2] Erwan Jahier, Nicolas Halbwachs, Pascal Raymond, Xavier Nicollin, and David Lesens. Virtual execution of AADL models via a translation into synchronous programs. In Christoph M. Kirsch and Reinhard Wilhelm, editors, *Proceedings of the 7th ACM & IEEE International conference on Embedded software, EMSOFT 2007, September 30 - October 3, 2007, Salzburg, Austria*, pages 134–143. ACM, 2007.

[3] Yue Ma and Jean-Pierre Talpin andThierry Gautier. Virtual prototyping aadl architectures in a polychronous model of computation. submitted, February 2008.

[4] Matthieu Moy. *Techniques and Tools for the Verification of Systems-on-a-Chip at the Transaction Level*. PhD thesis, INPG, Grenoble, France, December 2005.

[5] Ludovic Samper, Florence Maraninchi, Laurent Mounier, Erwan Jahier, and Pascal Raymond. On the importance of modeling the environment when analyzing sensor networks. In *3rd International Workshop on Wireless Ad-hoc and Sensor Networks (IWWAN'06)*, New York, USA, June 2006.