Synthesizable SystemC Code from UML Models

W.H. Tan[1], P.S. Thiagarajan[1], W.F. Wong[1], Y. Zhu[1] and S.K. Pilakkat[2]

[1] School of Computing, National University of Singapore, Singapore
[2] Institute for Infocomm Research, National University of Singapore, Singapore

1 Introduction

We present preliminary results concerning a translation that generates SystemC code from designs developed in UML using the Rational Rose RT tool. For suitably restricted designs, the generated SystemC code is synthesizable in that it is accepted by the Synopsis CoCentric compiler.

The construction of this translator is a part of our ongoing effort to understand design flows that would start with system descriptions using UML-notations and would produce full implementations of the software and hardware components as well as their communication interfaces. More precisely, we would like to study the intended role (if any) of a specific UML-notation (for example, deployment diagrams) and equally important, what might constitute a coherent combination of UML-notations in a design flow. Clearly, UML is an evolving standard especially due to the pressures brought about by the design needs of real time embedded systems [4][2]. However, significant parts of the UML have become stable and designers -especially software developers- will expect to use these notations in their initial specifications. It is these notations, such as class diagrams, state diagrams and structure diagrams that are initial focus of our study.

Since UML-based descriptions are quite far from implementations, it is clear that such descriptions need to be refined to a number of more detailed ones in order to obtain implementation level specifications. Here SystemC seems to be an excellent choice to serve as the layer immediately "below" the initial top-level UML-based design. SystemC is a design language that allows designs to be expressed and verified at sufficiently high levels of abstraction while at the same time enabling the linkage to hardware implementation and verification. Furthermore, SystemC, when viewed from a programming perspective, is a collection of class libraries built on top of C++ and so is naturally compatible with object oriented design methodologies. Therefore, creating a flexible and fully automated translation mechanism from the UML-based design language at the top layer to SystemC appears to be a promising strategy.

In the initial phase of our study, we decided to focus on an *existing* UML-based design language rather than formulating and advocating yet another system description language or model of computation. Since we wanted a working translation mechanism which can handle large and realistic examples, it was clear we should use an industrial-strength tool that would support the development of UML-compatible system descriptions. This led to the choice of Rose RT, which can produce C++ code for designs which involve just class diagrams, state diagrams and structure diagrams.

As mentioned earlier, we have at present a translator which can produce SystemC code from restricted Rose RT designs. The restrictions we place are not severe and they are explained in more details in the next section. Furthermore, the SystemC code that our translator produces is accepted by the Synopsis CoCentric compiler and in this sense, we also have a Rose RT wrapper for a synthesizable subset of SystemC. We now turn to a description of the main features of our translator.

2 The RT2SystemC Translator

Despite its intention as a tool for general purpose software development, Rose RT has close similarities to SystemC. A capsule in Rose RT corresponds to a module in SystemC. Capsules in Rose RT communicate via ports and protocols just as modules communicate via ports and channels in SystemC. A capsule undergoes a state transition when a specified trigger signal arrives whereas in SystemC this corresponds to an incoming signal on one of the ports specified in the sensitivity list of an SC_METHOD. Our translator exploits these similarities to identify and extract important sections of the C++ code generated by the Rose RT tool.

The fact that only class diagrams, state diagrams and structure diagrams are allowed at present is a limitaion of the Rose RT tool that generates C++ code . As for the restrictions imposed by our translator, they have to do with the fact that in the current first phase we would like to translate into the synthesizable subset of SystemC. Consequently, we require that at Rose RT level, attributes created in design must be accessed and modified only by the standard OO constructs, accessors and modifiers, respectively. The second restriction has to do with the user-defined C++ code and functions invoked in the Action section of a state transition in state diagrams. The sending and receiving of data associated with these functions are realized using the Rose RT functions `send()` and `getMsg()->getData()`. To facilitate the translation process, we require the designer to use these functions explicitly in the Actions section of the state transition. Our translator handles pointers by transforming them to synthesizable static allocations and the functions associated with the pointers are translated in a “call by value” style, where the associated allocations and function arguments are evaluated before they are accessed.

The schematic workflow of the translator is shown in Figure 1. The post-processing phase is needed only if synthesizable SystemC is the target. It adds hardware-dependent information such as the clock’s characteristics needed for synthesis. For TLM-level SystemC, this step can be eliminated. The various steps of the translation are automated by an executable PERL script which completes the translation in one go.

We have applied our translator to three design examples. The first is the sample Rose RT example called Ping Pong. We have also designed a simple slot machine dispensing drinks that consists of 10 capsules which handle various functionalities such as coin acceptance, coin return, drink selection, drink dispensing and machine control. In the last design example, we created a Rose RT version of the Simple Bus example given in TLM style by the Open SystemC initiative. Our translator produces synthesizable SystemC code for all three examples. Details of these example designs are available at: http://www.comp.nus.edu.sg/~zhuyx/UML2SystemC.html.

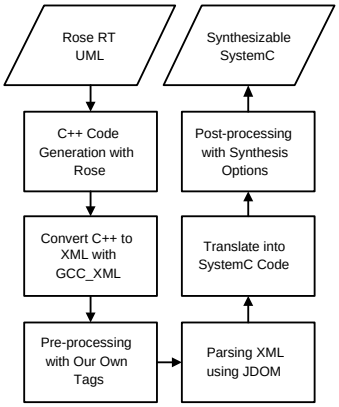

Fig. 1. The Translation Process

3 Summary

We have outlined here a translation mechanism that produces SystemC code from Rose RT designs. We are currently exploring ways of using our translator to design larger examples with the focus on translation into TLM-style SystemC. A similar exercise based on the Rhapsody tool is also almost complete. There are interesting similarities as well as differences between these tools as an UML-front-end for system designs and we hope to report on this comparison soon. A key line of research to pursue in the future will be verification techniques at both UML and SystemC levels. An equally, if not more, interesting line of enquiry concerns the generation of test suites. We expect HMSCs (High level MSCs) which are MSCs augmented with branching and iteration to play a fundamental role here. We feel that the development of powerful test suites at the UML-level and their translation and validation at the TLM-level of SystemC will contribute significantly to design, rapid prototyping and reuse of certifiably correct hardware and software co-designed components.

Due to the lack of space, we have not attempted to relate our work - and more significantly its context - to the large body of research in this field that has been reported in the literature. Briefly, we wish to point to the discussion in [4], the topics covered in [2] and the brief but focused survey in [3] as a much more elaborate discussion of the context of our work. As for the specific work on translation, related results are reported for instance in [1, 5] with the key difference being we did not add any new features to the UML-notation in our experiments.

References

1. F. Bruschi. A systemc based design flow starting from uml models. In *The 9th European SystemC users Group Meeting*, 2004.
2. L. Lavagno, G. Martin, and B. Selic. *Uml for Real: Design of Embedded Real-Time Systems*. Kluwer Academic Publishers, 2003.

3. G. Martin. Systemc and the future of design languages: Opportunities for users and research. In *The 16th Symposium on Integrated Circuits and Systems Design*. IEEE Press, 2003.
4. G. Martin, L. Lavagno, and J. Louis-Guerin. Embedded uml: A merger of real-time uml and co-design. In *The 9th Int'l Symp. on Hardware/software Codesign*, pages 23–28, March 2001.
5. Q. Zhu, A. Matsuda, and M. Shoji. An object-oriented design process for system-on-chip using uml. In *The 15th Int'l Symp. on System Synthesis*, pages 249–254. ACM Press, 2002.