

Interfacing UML 2.0 for Multiprocessor System-on-Chip Design Flow

Jouni Riihimäki
Nokia Technology Platforms
Tampere, Finland

Petri Kukkala, Tero Kangas, Marko Hännikäinen, Timo D. Hämäläinen
Tampere University of Technology
Tampere, Finland

Abstract—UML 2.0 can be extended for embedded system design. Our solution is a well-defined modeling approach, known as TUT-Profile, for UML 2.0 together with our System-on-Chip architecture exploration tools. The two major novel features are an explicit control of real-time constraints at UML level and the transformation of the original UML model using back-annotated results of SoC architecture exploration. In this way, all information is kept up to date in a single UML model, in contrary to other flows that use UML only as a front end. This paper focuses on the interface between UML model and the architecture exploration and presents conversions, tools, and intermediate format required for the flow.

I. INTRODUCTION

Use of a high-level language (HLL) in system-level design is a key to speed up the design process. The high abstraction level and possibility to describe both hardware (HW) and software (SW) in the same environment facilitate the management of complex designs. However, the existing lower level design tools, for example for logic synthesis or architecture exploration, do not typically support high-level languages such as Unified Modeling Language (UML), Esterel, or transaction level modeling properties of SystemC.

One solution to combine HLL with existing flow is to transform the HLL description to a form supported by an existing lower-level design flow. To provide support for multiple input languages and for several lower-level tools, a general intermediate representation for design data is required. Our solution is to use a custom XML-based system model.

UML 2.0 profile, called TUT-Profile [1], defines the description of application and architecture independent of each other as well as the mapping between them. The transformed UML models for application and architecture are given to architecture exploration tool called Koski [2], which optimizes the HW architecture and its parameters as well as the mapping to fulfill the requirements defined for example for real-time, area, and power. After that, the optimized design is back-annotated to the UML level and the UML design tool automatically visualizes the modifications. The updated information includes how the real-time requirements are met with the current architecture and application mapping.

Most of the existing UML-based design flows use UML only as a graphical front end that allows a designer to sketch the main parts of the design. However, they do not allow an automatic updating or transformation of the UML model according to the results of the architecture exploration. The automatic updating makes it possible to base the whole design flow to a single system description, which is always intact and up to date.

This paper is organized as follows. The next section discusses the related research. Section 3 introduces our UML 2.0 based design flow and Section 4 presents the interfacing tools to connect the UML development environment with Koski. A case study is presented in Section 5. The paper is concluded in Section 6.

II. RELATED RESEARCH

UML in system design is a widely researched area [3][4]. One of the major advantages of UML is that it allows using different semantics for different purposes. Several design flows from UML to hardware description languages are presented in [5], [6], and [7]. All of those generate SystemC description of the design. In [6] and in [7], a limited modeling style and restricted subset of UML is supported.

Similar design flows are also based on other languages. For example, in [8] and [9], the authors present design flows that start from Specification and Description Language (SDL) leading to VHDL models and further to physical implementations. The main difference between our approach and the previous research is the back annotation and automatic transformation of the UML model. Only [5] of the related works presents a link back to the UML level but does not allow architecture exploration.

In addition, several XML-based intermediate languages are presented in literature. For example, Y-Chart Modeling Language (YML) [10] used in Sesame [11]. YML is a very illustrative language for Y-Chart methodology of Sesame being able to be used to model complex functionality of the application. MOML [12] of Ptolemy project is another XML-based approach for system modeling. Also [5] uses a XML-based design data representation. It is targeted only for internal use of the presented tool. MOML is for general application modeling and simulation.

III. 3. UML 2.0 BASED DESIGN FLOW

Our UML 2.0 based design flow is presented in Fig 1. The key principle is to govern the design process in UML 2.0 by keeping all essential design information in application, architecture and mapping models. These models include also the constraints and control parameters for architecture exploration.

A UML 2.0 profile called TUT-Profile [1] defines the semantics for the embedded system design in UML. In TUT-Profile, the application is seen as a set of active classes with an internal behavior. The instances of active classes are called application processes. The processes communicate using signals, which are asynchronous events. Application modeling with TUT-Profile is presented in [13].

The overall design flow consists of an automatic C code generator, interfacing tool between UML 2.0 environment and Koski, Koski itself as an architecture exploration tool, and an implementation composer. Telelogic Tau G2 is used as a UML development tool and it is also used to generate C code. The implemented interface tool consist of an UML application parser, UML architecture and mapping parser, UML profiler, UML application performance back-annotator, and UML architecture and mapping back-annotator.

Koski architecture exploration tool [2] performs a two-phase optimization for the design. It consists of static and dynamic architecture exploration tools to optimize the structure and parameters of the HW architecture as well as mapping of application processes. The optimized architecture and mapping together with performance results are updated back to the UML level with the UML architecture and mapping back-annotator.

Finally, the Implementation composer creates a HW description based on optimization results utilizing components in the HW library. In addition, the generated application code is located to each processor in the HW

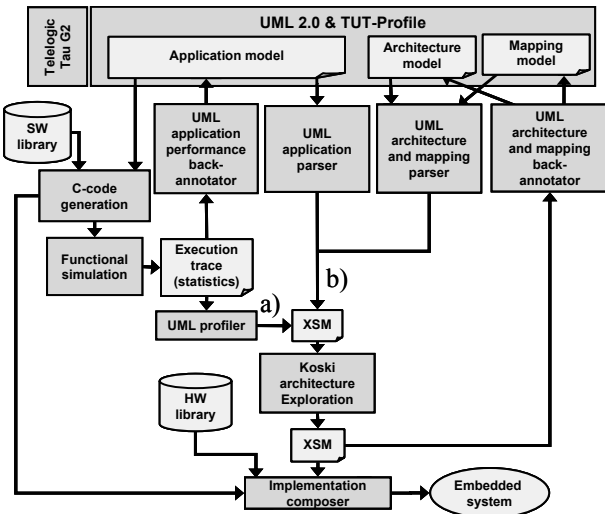


Figure 1 UML 2.0 based design flow.

```

<application>
  <process_network>
    <process pid="1" name="master:Master" ops="15" class="general">
      <input_function value="0 OR 1"/>
      <output_function value="TRUE"/>
      <out_port port_id="1" />
      <out_port port_id="2" />
      <out_port port_id="0" bytes="38016"/>
    </process>
    ...
    <connect_port input="1" output="9"/>
    <connect_port input="4" output="11"/>
    ...
  </process_network>
</application>

<platform>
  <hibi data_width="16" frequency="100000000"/>
  <agent>
    <wrapper id="1" address="N/A" prior="1" max_send="110"/>
    <pe id="1" name="CPU" type="ARM7" ops="1"/>
  </agent>
  <agent>
    <wrapper id="2" address="N/A" prior="2" max_send="110"/>
    <pe id="2" name="IO" type="HW_IO" ops="2"/>
  </agent>
</platform>

<mapping>
  <mapping pid="1" pe_id="2" type="fixed" />
  <mapping pid="2" pe_id="1" type="fixed" />
  ..
</mapping>

```

Figure 2 An example design in XSM format.

architecture. The implementation consists of several phases including logic synthesis from the generated RTL model.

It should be noted from Fig 1 that there are two alternative paths from the UML application model to the utilized intermediate format called XSM. Dynamic method (a) is based on simulation and in static approach (b) the UML design is statically analyzed for optimization. The first one uses code generator and UML profiler whereas the latter approach utilizes UML application parser.

Telelogic Tau G2 saves a UML design to a XML file. In addition to the actual UML model, the file contains additional information, for example UML project data and coordinates for the graphical representation. The hierarchy of the file is very deep. Therefore, the original file format is not used directly in our flow. Instead, a new tidy and more compact representation called XML system model (XSM) is designed. The optimized design is also stored to XSM, which is then used to update the UML model. XSM is also used when constructing a final architecture.

TABLE I XSM PARAMETERS

| Element | Available parameters |
|-----------------|---|
| Process | Complexity (ops), Id, Name, Process type, Max execution time, Input function, Output function, Transfer target, Data size |
| PE | Frequency, Id, Name, Type, Performance (ops per cycle) |
| Wrapper | Address, Priority, Id, |
| Interconnection | Frequency, Type, Name, Data width, Transfer capacity |
| Mapping | Process ID, ID of target PE, Mapping type (Fixed/Optimizable) |

Fig 2 shows an example of XSM. The application in XSM is modeled as a Kahn process network (KPN). Architecture and mappings can be modeled similarly as the application, as shown in Fig 2. TABLE I introduces the parameters that are used in XSM description. However, since XSM is in XML format, user can declare new parameters

when needed. The structure of XSM is close to YML [10] but the application modeling capabilities are more limited in XSM. However, XSM is developed to be used with our flow and therefore only the required structures are included to it.

IV. INTERFACING UML 2.0 AND KOSKI

The implemented UML interface tool consists of five TCL scripts, consisting of the parsers and back-annotators depicted in Fig 1. The interface tools expect that the UML models are saved in a Telelogic Tau G2 compatible file format. The tools are modular and only the module that handles the UML file needs to be replaced for other UML development tools. The interface scripts and their implementation sizes are shown in TABLE III

The UML application parser converts the application model to KPN statically. It does not consider the actual functionality of the application processes. Instead, it recognizes the assigned tagged values that indicate the required execution time for each application process.

The functionality of the parser is straightforward. It goes through the saved XML tree and recognizes UML classes, class instances, signals, and dependencies between different objects. The meaning of each object is recognized based on the assigned TUT-Profile stereotype. Finally, the created application model is stored to XSM. This static application parsing requires that TUT-Profile is strictly followed.

An alternative way to generate the application part of XSM is to use the UML application profiler. For this, we use the automatic C code generator to create an executable application model. SW library in Fig 1 contains custom profiling functions that can be integrated to the generated application code. These functions generate execution trace for the simulated application. The execution time of the profiling functions is typically short and can be neglected. The functions can be utilized to profile any application.

The application code is instrumented with profiling functions to trace the execution. The code generation of the employed Telelogic Tau G2 contains tracing facilities, which allow to use custom functions that are stored to the SW library. The trace contains information of state transitions inside application processes, including the triggering signal and time stamps for state transitions. In addition, the target and source process of each signal exchange are also stored. The trace is also in XML format as shown in Fig 3.

In this example, process 11 is triggered by the signal from the process 12. Also the amount of received data is indicated. First, the process is at state 2 and it sends signal 24 to process 14, and ends to state 2. After that, the signal 24 triggers the execution of process 14. Time stamps are included to all activities.

The UML profiler converts the trace to XSM. It recognizes the application processes from the execution trace. In addition, it finds the dependences between processes. The time stamps are used to determine the execution time for each process. These metrics are back

annotated to application model with the UML application performance back-annotator to be used later in the architecture exploration.

The execution trace can be obtained from a functional simulation on the workstation as shown in Fig 1. Alternatively, the execution trace can be generated during an execution performed on the real target platform. The execution trace is dependent on the applications input sequences, and thus, it may lead to an incomplete process network. However, profiling is quite often the only method accurate enough, especially in reactive applications such as telecommunications protocols. In these, the execution is highly conditional by nature, and cannot be natively modeled using a pure dataflow MoCs.

In addition to the application model, the user describes the HW architecture and controls the mapping of the application processes to the defined architecture components. The architecture and mapping models are optional; they are not required by the Koski architecture exploration, but if the information exists, it is used to guide the optimization. The UML architecture and mapping models are statically parsed and given to Koski in the XSM format using the UML architecture and mapping parser. Functionality and structure of those parsers are similar to the UML application parser.

In our flow, there are two tools to modify the original UML models. The performance results from simulations, i.e. measured execution times for each application processes, are back annotated with the UML application performance back-annotator. This tool does not change the structure or functionality of the application.

Another tool is for updating the optimized architecture and mapping from Koski. This is done with the UML architecture and mapping back-annotator. This tool automatically updates the modifications made to the original UML model. Therefore, UML models are always kept consistent to store all essential design information. Moreover, the results of the architecture exploration, and the information on how the set requirements are met, are shown to the designer.

The back-annotation allows a designer to observe the modifications in the UML level, which facilitates the management of the design. In addition, the designer can easily modify the design further, and if necessary, override the performed modifications.

```

...
<transition process="11">
  <trigger signal="1" sender="12" data_length="128" />
  <start state="2" time="96.392478300" />
  <output signal="24" time="96.392415490" receiver="14" />
  <end state="2" time="96.392783070" />
</transition>

<transition process="14">
  <trigger signal="24" sender="12:0" data_length="48" />
  <start state="1" time="96.392879960" />
  <output signal="24" time="96.392811260" receiver="4" />
  <end state="2" time="96.392949240" />
</transition>
...

```

Figure 3 An example of the execution trace file.

V. CASE STUDY

The test case is a Medium Access Control (MAC) protocol, which is a part of a Wireless Local Area Network (WLAN) terminal called TUTWLAN [14]. Several time critical functions based on real-time requirements can be identified from the TUTMAC protocol, for example Cyclic Redundancy Check (CRC) of TUTMAC frames. The time critical functions need time bounded execution in order for the TUTMAC protocol to meet the delay requirements when reacting to received and transmitted frames.

TUTMAC is implemented utilizing UML 2.0 and the implementation consists of over twenty application processes. The initial platform consists of two NIOS II processor cores, HW accelerator for CRC32, and a radio interface. The modules are connected with HIBI v2 network-on-chip (NoC) [15]. The design is targeted to Altera Stratix FPGA.

The UML application profiler and UML architecture and mapping parsers are used to create an XSM description of the design, which is given to Koski. TABLE II shows the line count of original UML files, profiling data, and created XSM file. Simulation length affects to the size of profiling log and application affects to the required simulation length, since all different cases should be profiled. Created XSM file is about 0.6% of the UML files size. The application model is not touched during optimization, and the size of architecture and mapping model is roughly the same also after the optimization results are updated to the UML model.

TABLE II AMOUNT OF DESIGN DATA IN DIFFERENT FORMATS.

| File | Lines |
|---|----------|
| UML Application model | 47,000 |
| UML Architecture & mapping model | 4,300 |
| Profiling data (10 simulated seconds) | ~100,000 |
| XSM file (application, HW, and mapping) | 320 |

When the test case was executed on a standard laptop with 1.7GHz Pentium processor and 1GB of main memory running Windows XP and Cygwin, the executing times presented in TABLE III was measured. TABLE III summarizes also the implementation size of each tool.

TABLE III IMPLEMENTATION SIZE AND EXECUTION TIMES OF THE INTERFACE TOOLS.

| Tool | TCL code lines | Execution time [sec] |
|--|--------------------------------------|----------------------|
| UML application parser | 2,000 | 15 |
| UML architecture and mapping parser | 1,500 | 10 |
| UML profiler | 1,000 + 100 lines of C in SW library | N/A |
| UML architecture and mapping updater | 2,500 | 80 |
| UML application performance back-annotator | 1,500 | 20 |

VI. CONCLUSION

The presented flow and the UML interface tools connect UML 2.0 development environment to the lower-level

architecture exploration tool called Koski. The model transformation between UML and Koski domains is enabled by the TUT-Profile and the XML system model (XSM). The automatic updating of the UML model keeps the design synchronized with the results of the optimization tools. In addition, UML visualizes the performance optimizations to the designer in a graphical form.

The static UML parser can be used to find an accurate structure of the UML application model that meets the rules given in TUT-Profile. The UML Profiler, in turn, facilitates determining the structure of any application. The presented results show the usability of the UML interface tools and the whole flow. In addition to XSM conversion, a link from UML to SystemC and back to UML is preferred to enable also HW synthesis.

REFERENCES

- [1] P. Kukkala et al., "UML 2.0 Profile for Embedded System Design," Proceedings of Design, Automation, and Test in Europe (DATE'05), 2005, pp. 710-715.
- [2] T. Kangas et al., "A Communication-Centric Design Flow for HIBI-based SoCs", LNCS 3133 Computer Systems: Architectures, Modeling, and Simulation, A.D. Pimentel, S. Vassiliadis, (eds.), Springer-Verlag, Berlin, 2004, pp. 474 - 483.
- [3] L. Lavagno, G. Martin, and B. Selic, UML for Real: Design Embedded Real-Time Systems, Kluwer Academic Publishers, 2003.
- [4] UML – SoC Homepage, <http://www.c-lab.de/uml-soc/>
- [5] K.D Nguyen et al., "Model-driven SoC Design via Executable UML to SystemC," Proceedings of 25th IEEE International Real-Time Systems Symposium (RTSS), Dec. 2004, pp. 459-568.
- [6] W.H. Tan et al, "Synthesizable SystemC code from UML models," In UML-SoC workshop in DAC'04 conference. Available in http://www.comp.nus.edu.sg/~ctp/publications/UML_SoC.pdf.
- [7] Z. Sun et al, "Design of Clocked Circuits using UML", Proceedings of the Asia and South Pacific Design Automation Conference 2005 (ASP-DAC), 2005, pp. 901-904.
- [8] W. Horn et al., "Hardware Synthesis of an ATM Multiplexer from SDL to VHDL: A Case study," Proceedings of IEEE Computer Society Workshop on VLSI, 1999, pp. 100-105.
- [9] C. A.M. Marcon et al, "Modeling of Embedded Digital Systems from SDL Language: a Case Study," XVII SIM – South symposium on Microelectronics, Brazil.
- [10] J. E. Coffland et al, "A Software Framework for Efficient System-level Performance Evaluation of Embedded Systems", Proceedings. of the 18th ACM Symposium on Applied Computing, Embedded Systems track, Melbourne, Florida, 2003, pp. 666-671.
- [11] SESAME - Simulation of Embedded Systems Architectures for Multi-level Exploration, <http://sesamesim.sourceforge.net/>
- [12] E. A. Lee and S. Neuendorffer. "MoML - a Modeling Markup Language in XML," version 0.4. Technical Report UCB/ERLM00/8, Electronics Research Lab, University of California.
- [13] P. Kukkala et al., "UML 2.0 Implementation of an Embedded WLAN Protocol". Proceedings of the 15th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2004), 2004, Spain, pp. 1158-1162.
- [14] M., Hännikäinen et al, "TUTWLAN – QoS supporting wireless network," Telecommunication Systems - Modelling, Analysis, Design and Management, 2003, pp. 297-333.
- [15] E. Salminen et al., "HIBI v.2 Communication Network for System-on-Chip," LNCS 3133 Computer Systems: Architectures, Modeling, and Simulation, A.D. Pimentel, S. Vassiliadis, (eds.), Springer-Verlag, Berlin, 2004, pp. 412 - 422..