SysML and Systems Engineering Applied to UML-Based SoC Design

Yves Vanderperren, Wim Dehaene

Katholieke Universiteit Leuven,
EE Department (ESAT-MICAS)

Abstract. UML is gaining increased attention as a system design language. This is confirmed by several reported experiences and current standardization activities such as the SysML initiative, which extends UML toward the Systems Engineering domain. This paper summarizes the main features of SysML and analyzes the synergies between Systems Engineering and complex SoC design using UML. In particular, the application of Requirements Engineering is illustrated in the SoC context, and suggests possible improvements to existing SoC design processes based on UML.

1 Introduction

UML is attracting growing interest as a system level visual language to support the tasks of analyzing, specifying, designing, verifying and validating systems-on-a-chip (SoC) [1]. Indeed UML presents the particular advantage of allowing customization for specific application domains, such as SoC design, while providing unification [2]:

- *across historical notations*: UML 2 is the result of a long evolution from the 'method wars',
- *across development life cycles*: despite its primary focus at system level, UML can be used in other phases of the SoC development lifecycle, such as prototype test [3, 4],
- *across application domains*: UML has the capability of providing a unified view of a heterogeneous SoC,
- *across development processes*: UML can be associated with any particular design method of choice,
- *across design languages*: several synergies between UML and existing SoC design languages at different abstraction levels have been reported in the recent years, such as UML/SystemC [5–7] or UML/VHDL [8, 9].

However the semi-formal aspect of UML and its lack of clear semantics constitute a well-known issue. Moreover, UML still has strong roots in the software domain despite its official and domain-neutral definition [10]. SysML [11] is an extension of UML which allows modeling a system from a Systems Engineering (SE) point of view. Strong similarities exist between the methods used in the

In: Proc. 2nd UML-SoC Workshop at 42nd DAC, Anaheim (CA), USA, 2005

area of SE and complex SoC design, such as the need for precise requirements management, heterogeneous system specification and simulation, system validation and verification. This paper gives an overview of SysML and identifies the opportunities to improve UML-based SoC development processes with the successful experiences from the SE discipline.

2 SysML: UML for Systems Engineering

The Systems Modeling Language (SysML) is the result of a joint initiative of OMG and the International Council on Systems Engineering (INCOSE). SysML is a (draft) profile extending the UML 2.0 Superstructure Specification and provides a standard modeling language to support the specification, analysis, design, verification and validation of a broad range of complex systems which are not necessarily software based.

SysML is based on the minimal subset of UML that satisfies the needs of systems engineers, and adapts UML only when it is necessary. One of the major improvements SysML brings to UML is the support for representing requirements and relating them to the model of a system, the actual design and the test procedures. Indeed UML does not address how to trace the requirements of a system from informal specifications down to the individual design elements and test cases, although this activity is crucial. Use Cases help build up a sound understanding of the expected behavior of the system and validate the proposed architecture. However requirements are often only traced to the use cases but not to the design. Adding design rationale information capturing the reasons for design decisions made during the creation of development artifacts and linking these to the requirements helps to analyze the consequences of a requirement change. SysML therefore introduces the *requirement diagram* and defines several kinds of relationships for improving the requirement traceability. The purpose is not to replace the available commercial tools dedicated to this subject, but to provide a standard way of linking the requirements to the design and the test suite within UML. Requirements can be decomposed by means of the *containment* relationship in a similar way to that for class diagrams. The *trace* dependency relates derived requirements to source requirements. The system designed and the requirements are linked by a *satisfaction* dependency. Finally, the *verification* dependency associates a requirement with the test case used to verify this requirement. Requirement diagrams contribute to the rigorous transfer of specifications and related information among tools used by systems, software and hardware engineers.

SysML enhances the UML diagrams used to represent the structural aspects of a system. It introduces the concept of *assembly*, a stereotyped class which describes a system as a structure of interconnected parts. An assembly provides a domain neutral modeling element that can be used to represent the structure of any kind of system, regardless of the nature of its components. In the context of a SoC, these components can be hardware or software based as well as analog or digital. An *assembly model*, the SysML version of the composite structure

diagram, shows the interconnection of the parts of a system and supports information flows between components, as present for example in the datapath of a heavy signal-processing oriented SoC.

The concept of *allocation* in SysML is a more abstract form of deployment than in UML. It is defined as a design time relationship between model elements which maps a source into a target. An allocation provides the generalized capability to allocate one model element to another. For example, it can be used to link requirements and design elements, to map a behavior into the structure implementing it, or to associate a piece of software and the hardware deploying it.

As a consequence, SysML abandons the use of the communication diagram. The SysML support for flows between assemblies is meant to provide an equivalent and domain-neutral modeling capability. The same argument applies to the deployment diagram. It is considered too specific to the software domain as it is limited to the deployment of software artifacts onto hardware components, while the SysML allocation provides a more flexible deployment mechanism.

Finally, SysML provides several enhancements to activity diagrams. In particular the control of execution is extended such that running actions can be disabled. In UML 2 the control is limited to the determination of the moment when actions start. In SysML a behavior may not stop itself. Instead it can run until it is terminated externally. For this purpose SysML introduces control operators, i.e., behaviors which produce an output controlling the execution of other actions.

These new features from SysML can be applied to SoC design [12]. SysML provides therefore a bridge between the domains of Systems Engineering, UML, and SoC design. This important point needs to be carefully understood, however:

- SysML does not solve the question of the lack of semantics in UML. SysML does also not define always precise semantics for the modeling elements it introduces: although the concept of assembly allows unifying the representation of heterogeneous systems, this achievement comes at the cost of increased abstraction and weaker semantics. Abstraction is a double-edged sword since it is necessary in order to deal with increasing complexity but can cause interpretation issues.
- When considering the particular aspects of complex SoCs, the ability to navigate both horizontally and vertically through the system architecture is of major importance. SysML allows the integration heterogeneous domains in a unified model at a high abstraction level. The semantic integrity of the model of a heterogeneous SoC could be ensured if future tools supporting SysML take advantage of the concept of allocation and provide facilities to navigate through the different abstraction layers into the underlying structure and functionality of the system.
- SysML, like UML, is neither a methodology nor does it dictate any particular development process to be used. SysML provides systems engineers with modeling means to enhance the communication of the design intent, but it does not tell how these can be best applied within a design flow. This is

the role of a development process, the backbone of any project based on UML/SysML.

This last comment is of paramount importance and is further detailed in the remainder of this paper. Several synergies can indeed be identified between the disciplines involved in Systems Engineering, complex SoC design flows, and aspects related to the definition of a design process based on UML/SysML.

3 Systems Engineering and UML/SysML-based SoC design processes

INCOSE [13] defines Systems Engineering as an "*interdisciplinary approach* and means of enabling the realization of successful systems. It focuses on defining *customer needs and required functionality* early in the development cycle, documenting requirements, then proceeding with design synthesis and system validation while considering the complete problem. Systems engineering integrates all the disciplines and speciality groups into a team effort forming a structured *development process* which proceeds from concept to production to operation.". This definition is analyzed in the context of SoC design in the following sections.

Systems Engineering as an interdisciplinary approach. Fabless system providers generally lack expertise in designing state-of-the-art submicron VLSI circuits. On the other hand, silicon providers often do not have the global visibility and the overall system knowledge required to define a complex SoC. However, the successful construction of such systems requires a cross-functional team with system design knowledge combined with an experienced SoC design team from hardware and software domains [14].

The Essential Role of Requirements Engineering. Requirements Engineering (RE) constitutes a branch of Systems Engineering which must span the gap between the informal world of the stakeholders needs and the formal world of the system's behavior. RE for SoC can be performed in different contexts, such as market-driven product development (ASSP) or for a specific customer (ASIC, FPGA), and leads to a particular choice of possible *requirements elicitation techniques* (see figure 1) [15]. Four of these techniques suit especially the domain of RE for SoC using UML/SysML. Model-driven SoC design flows, where an executable model of the system is built early in the project lifecycle using an electronic system languages (ESL), share similar principles as the *prototyping* requirements elicitation technique. Executable UML could support new approaches at system level, in particular for control-oriented and software-dominated embedded systems. *Goal-oriented* techniques [16] stress the question why a system is needed, regardless of the details of how to achieve the goals. These correspond to the services to be provided (functional concerns) as well as the required quality of these services (non functional concerns, typically throughput, execution time

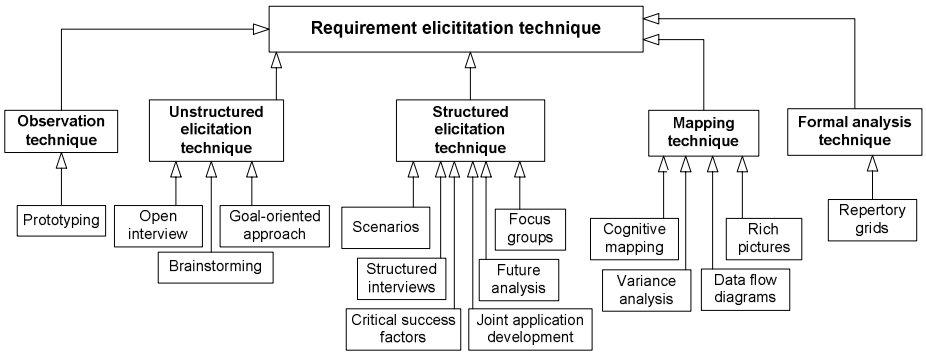

Fig. 1. Requirements Elicitation Techniques

and power consumption in the SoC context). Such techniques can be applied in conjunction with approaches structuring use cases with goals [17, 18]. Structured elicitation techniques based on *scenarios* and mapping techniques based on *data flow diagrams* are supported by the behavioural diagrams available in UML/SysML and can be applied to SoC design. Scenarios generated by any of the previous techniques help to validate the SoC requirements and verifying their completeness. The object-oriented aspects of UML are actually not required at this stage, where the system is seen as a black box, and several techniques from the structured systems development and Systems Engineering methodologies are applicable.

Defining requirements not only involves the task of requirements elicitation, but also the *modeling of requirements*, which is concerned with the efficient representation of the accumulated information. The SysML requirements diagram fills here the gap of UML.

Besides the requirements definition, managing *change and evolving requirements* is another fundamental RE activity. This task calls for cross-compatible tools which can exploit the standard SysML traceability links between requirements as well as from requirements to design and test, in order to facilitate the analysis of the consequences of a requirement change. Requirements can not only evolve in time, but across product families as well. Platform-based designs [21, 22] can benefit substantially from the experience of RE in this domain, as they need a dedicated attention to the definition of core and differentiating requirements. Different platform instances share indeed similar requirements and architectural characteristics, but differ in certain key requirements depending on the target application.

Managing *inconsistencies* is an essential concern when defining requirements. Viewpoint modeling [19] can be used not only as a requirements validation method, but also to organise a multi-perspective SoC development technique so that the specification process remains a coordinated effort. Viewpoints allow constructing the description of large and complex systems from different perspectives and are supported by SysML. Examples of possible applications in the

SoC domain include the verification of a hardware-software interface, or the integration of an intellectual property (IP) block within a system. In these cases, it is essential to focus not only on specifying the functionality of the (sub)system under consideration, but also on modeling the properties of the environment and what the (sub)system should achieve in this environment. This is one of the motivations for goal oriented techniques, which decrease the emphasis on modeling information flow and system state, and concentrate instead on capturing the system's purpose within its context.

Systems Engineering and SoC Development Processes. It is crucial to adapt the development process to the particular needs of the given application domain, much the same way as UML requires customization toward the area in which it is used. In the SoC context, characterized by severe first time right requirements and exponentially increasing mask costs which prevent successive physical implementations, executable models provide a means to support iterative development processes. As an example, a customized version of the RUP [20] was applied in [14]. The importance of a system model makes SoC development processes based on UML [14, 18, 21, 23] resemble Systems Engineering processes, such as SIMILAR [24] or HARMONY [25]. As illustrated by the experience reported in [3] compared to [23], re-evaluation and continuous improvement of the development process, one of the most important phases of SIMILAR, plays a crucial role in the successful integration of UML into SoC design processes. Moreover, the reported experience indicated that 55% of the errors detected were related to inconsistencies in the specification during the analysis and modeling phase, while another 21% were discovered during to black box implementation tests based on scenarios derived from use cases. These results confirm the benefits of a clear requirements management procedure and the applicability of UML at different phases of the design flow.

These SoC development processes based on UML could be improved by integrating further the requirements management as a lifecycle task running in parallel with the design and verification activities, as recommended by the underlying principles of the Requirements-Based UML (RBU) process [26]. Another aspect which deserves more attention concerns the investigation of design alternatives, as stressed in a dedicated phase of SIMILAR. This task requires the definition of clear performance and cost figures of merit, and the possibility to proceed fast with the design synthesis of the alternatives. The former might however be particularly difficult to estimate accurately at system level, in particular power related aspects, while the latter motivate programmable and reconfigurable architectures as well as improved support for automatisation.

4 Conclusions

The recent advances in UML and SysML present a valuable milestone for the application of UML to modern chip design. However, it is essential to remember that SysML, like UML, only provide standard modeling means. Neither UML

nor SysML solve the difficulty associated with systems analysis, but they allow efficient communication between the project stakeholders. A sound development process which suits the peculiarities of SoC design is necessary to complement the use of UML/SysML for SoC design. Regardless of the specific characteristics of the particular adopted process, it is beneficial to extend SoC engineering from its focus on hardware and software technologies to an engineering discipline of system development. A multi-disciplinary approach is indeed crucial to coping with increasing product complexity and the need for reduced design time. In particular, SoC designers can capitalize on the experience accumulated in the domain of Requirements Engineering in order to improve substantially product quality and design productivity.

References

1. Martin, G., Müller, W. (Eds.): UML for SoC Design. Springer, 2005.
2. Holt J.: UML for Systems Engineering. Institution of Electrical Engineers, 2004.
3. Zhu, Q., Oishi, R., Hasegawa, T., Nakata, T.: Integrating UML into SoC Design Process. Proc. Design, Automation and Test in Europe (DATE), 2005, pp. 836–837.
4. Vanderperren, Y., Dehaene. W.: UML for SoC: Added Value or Hot Topic?. Tutorial. DAK-forum, 2004.
5. Vanderperren, Y., Pauwels, M., Dehaene, W., Berna, A., Özdemir, F.: A SystemC Based System On Chip Modelling and Design Methodology. In "SystemC: Methodologies and Applications", Chapter 1, Mueller, W., Rosenstiel, W., Ruf, J. (Eds.), Kluwer Academic Publishers, 2003.
6. Baresi, L., Bruschi, F., Di Nitto, E., Sciuto, D.: SystemC Code Generation from UML Models. In "System Specification & Design Languages", Chapter 13. Kluwer Academic Publishers, 2003.
7. Nguyen, K.D., Sun, Z., Thiagarajan, P.S.: Model-Driven SoC Design Via Executable UML to SystemC. Proc IEEE International Real-time Systems Symposium (RTSS), 2004.
8. Björklund, D., Lilius, J.: From UML Behavioral Descriptions to Efficient Synthesizable VHDL. Proc. IEEE Norchip Conference, 2002.
9. McUmber, W.E., Cheng, B.H.C.: UML-Based Analysis of Embedded Systems Using a Mapping to VHDL. Proc. IEEE International Symposium on High-Assurance Systems Engineering (HASE), 1999, pp. 56-63.
10. OMG: UML 2.0 Infrastructure Specification. http://www.omg.org.
11. SysML Forum: SysML Specification (Draft), 2005. http://www.sysml.org
12. Vanderperren, Y., Dehaene, W.: UML 2 and SysML: An Approach to Deal with Complexity in SoC/NoC Design. Proc. of the conference on Design, Automation and Test in Europe (DATE), 2005, pp. 716–717.
13. INCOSE: What is Systems Engineering? http://www.incose.org
14. Vanderperren, Y., Sonck, G., Van Oostende, P., Pauwels, M., Dehaene W., Moore T.: A Design Methodology for the Development of a Complex System-on-Chip Using UML and Executable System Models. Proc. Forum on Specification and Design Languages (FDL), 2002.
15. Darke, P., Shanks, G.: User Viewpoint Modelling: Understanding and Representing User Viewpoints During Requirements Definition. Information Systems Journal, **7**:3, 1997.

16. van Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Guided Tour. Proc. 5th IEEE International Symposium on Requirements Engineering, 2001, pp. 249-263.
17. Cockburn, A.: Use Cases, Ten Years Later. Software Testing and Quality Engineering Magazine (STQE), **4**:2, 2002, pp. 37-40.
18. Vanderperren, Y., Dehaene, W.: A Model Driven Development Process for Low Power SoC Using UML. In [1], Chapter 10.
19. Easterbrook, S., Nuseibeh, B.: Managing Inconsistencies in an Evolving Specification. Proc. 2nd IEEE International Symposium on Requirements Engineering, 1995, pp. 48–55.
20. Kruchten, P.: The Rational Unified Process: An Introduction (3rd Ed.). Addison-Wesley, 2003.
21. Green, P. N., Edwards, M. D.: Platform Modelling with UML and SystemC. Proc. Forum on Specification and Design Languages (FDL), 2002.
22. Sangiovanni-Vincentelli A., Martin, G.: Platform-Based Design and Software Design Methodology for Embedded Systems. IEEE Design and Test of Computers, **18**:6, 2001, pp. 23–33.
23. Zhu, Q., Matsuda, A., Kuwamura, S., Nakata, T. , Shoji, M.: An Object-Oriented Design Process for System-on-Chip Using UML. Digest of IEEE Int. Solid-State Circuits Conf., 2002, pp. 249-254.
24. Bahill, A.T., Gissing, B.: Re-evaluating Systems Engineering Concepts Using Systems Thinking. IEEE Trans. Systems, Man, and Cybernetics–Part C: Applications and Reviews, 1998, pp. 516-527.
25. Hoffmann, H.-P.: UML 2.0-Based Systems Engineering Using a Model-Driven Development Approach. I-Logix White Paper, 2004.
26. Schulz, J.: Requirements-based Unified Modeling Language. Borland White Paper, 2003.