System Level Design Methodology

P.H.A. van der Putten, J.P.M. Voeten, M.C.W. Geilen and M.P.J. Stevens
Section of Information and Communication Systems
Eindhoven University of Technology, The Netherlands
piet@ics.ele.tue.nl

Abstract

There are many fundamental problems in the design of object-oriented methods that support the development of formal executable models on a system level, and that are suitable for hardware/software co-specification. System level description formalisms should combine concepts expressive enough to model the essentials of a system on the right level of abstraction.

This paper reports experiences in developing a specification and design method SHE (Software/Hardware Engineering) which is based on a formal language POOSL (Parallel Object-Oriented Specification Language). The method offers a path from an informal specification to a unified formal model that enables evaluation of system properties. This paper describes concrete new results as well as an approach towards research on system level methodology.

1. Introduction

A System-level-oriented design approach will become a key issue for successful development of hardware/software in future industrial systems and consumer products. The application of a collection of embedded processor cores in products is a reality in many products now. These products can be characterised as real-time, distributed, parallel, reactive systems. These characteristics make their design very complicated. Integration of such systems on a chip confronts us with all the problems of system design on top of the difficulties in VLSI design itself.

In the past, custom integrated circuits were subsystems whose requirements became apparent when the system specification and design process was evolving such that much knowledge could have been gathered already. A system on a chip must be designed by a risky direct step from requirements to implementation. Fortunately the flexibility of integrated reconfigurable subsystems, reconfigurable software and switch matrices will be available to solve particular specification errors. However the tremendous complexity together with the short path from specification to implementation will force the VLSI community to use structured specification methods and abstract system models to explore conceptual solutions before implementation.

The problems of system level design and board design migrate to VLSI design. Therefore the partitioning of a system will be influenced by many more criteria than addressed by the current research on co-design. This requires an extended multidisciplinary approach on top of the co-design approach. Whether products are successful will strongly depend on the selection of the right properties and functionality for the right market segment.

Currently system level design is not well supported in industry [1]. This hinders verification and validation of system properties and functionality. These activities should begin as early as possible in the development process. This paper reports on various aspects of the design of a system level specification and design method with a focus on methodology. Section 2 describes the role of formal semantics for system level methodology. Section 3 describes the activities and the design phases related to human communication. Section 4 illustrates the conceptual approach towards method design. A brief concrete overview of the system level specification and design method Software/Hardware Engineering(SHE), and the underlying formal object-oriented specification language (POOSL) is presented in Section 5. Section 6 summarises our approach towards method design and identifies important fields of system level methodology research.

2. Method design approach

There are various approaches for filling the gap between specification and sub-system design and implementation. Many object-oriented software analysis methods are developed for programmers needs. The opposite way is to create a method based on the needs for analysis, followed by a mapping from specification to implementation. This approach can offer flexibility to various forms of implementation. However this flexibility should not be based on a lack of formality. Much of the

praised flexibility in analysis methods has to do with their weakly defined semantics. Flexible human interpretation is used to bridge the gap between specification and synthesis. This is not the way to produce predictable and reproducible results from system level descriptions. A system level specification should be formal enough to be executable. Creating a formal mathematical semantics is a safe way to enable this. It is also a perfect basis for verification tool development.

The SHE method is the result of research on concepts for system level co-specification and design. We consider our approach [2] as long term research, in contrast to more directly applicable approaches that are built on currently available standards and tools [3], [4].

The core of the SHE method is the specification language POOSL which is based on formal mathematical semantics. Building a method and a language is not a trivial task. There are concepts that are orthogonal and that can be combined, however there are also many conflicting or mutually dependent concepts. This does not become apparent without in depth reasoning and understanding necessary to create formal semantics. Confusion about differences between existing methods is caused by the resemblance of graphical notations and languages statements, and the lack of understanding that crucial differences are in the semantics of underlying concepts.

All concepts such as real-time, concurrency, communication, sharing, hiding, etcetera, should be defined after exploration of alternative interpretations. The selection of blending concepts yields compositional and expressive language primitives. Investigation of all possible combinations of primitives in meaningful behaviour descriptions is necessary to accomplish this.

Besides the challenge of the construction of a system level description language, a connection between the system level description language, synthesis tools and verification tools must be realised. The feasibility of this connection is strongly influenced by the semantics of the chosen concepts and primitives on both sides. For synthesis a direct automatic mapping is preferred. For verification it must be possible to draw adequate abstractions from the models.

3. System level analysis

The essence of system level specification is the combination of system level architecture design with requirements capturing and functional specification. Design and specification cannot be separated [2]. Further the specification of a complex system requires description of both system and environment. In practice it is impossible to define the functionality of a complex system without creating a model. Such a model contains information about objects, about information to be processed, about various forms of communication, about logical, physical and interconnection structure, order of activities and many other properties such as real-time and performance properties. A model is a tool to manage complexity for individual designers and to enable multidisciplinary discussions.

A system level model should be a complete specification in the sense that it should contain all system properties that can be modelled by the chosen formalism. We have a long way to go before we can formalise all relevant system properties in a system specification process. For the time being, the human role remains dominant. First time right design requires further improvement of the possible ways for validation and verification.

Human communication will remain the most difficult part to get hold of. Mutual misunderstandings between (non)professionals based on different interpretations should be removed. Formal semantics can produce right interpretations, or reveal inconsistencies in languages or methods.

We propose to partition a specification and design process into phases. The first phase is to gather initial requirements, purpose descriptions and prescribed technologies based on a particular product idea (preparation phase). This enables deciding on an investment in developing a system specification (model) using a structured approach.

In the second phase an essential model is created. This integrates architecture structure, domain analysis, functional analysis, and system level property specification. This model is abstract enough to enable communication with a wide variety of experts, users, etcetera. Main goal of this phase is selecting a successful system level solution.

The third phase incorporates decisions about implementation technology into the model. The model is extended with objects that realise additional behaviour (interface technology, communication protocols, etcetera). The extended model is meant for communication among design engineers. Main goal is to prepare the actual software and hardware design phase (fourth phase). The SHE method supports the development of the essential model and the extended model. The other phases are not addressed in this paper.

Rigorous verification and validation of a system specification requires executable models. So in both the essential and the extended phase there are two activities. Firstly a model must be created that is as complete as possible. Secondly this model must be verified, validated and improved during subsequent iterations. The investigation of identified architectural and performance problems can be accomplished by developing separate

abstract models that address particular properties of concern.

4. Formally based object-oriented analysis

Traditional object-oriented analysis methods emphasise the search for classes and their relationships. More modern methods add analysis of collaborating instances, system structure and dynamic behaviour. Examples are ROOM [5] and UML [6]. These methods use various representations to analyze collaborations of instances, system structure and dynamic behaviour. ROOM aims at development of pseudo-concurrent software systems and UML at software systems and business modeling.

The SHE method aims at Hardware/Software co-specification. Although SHE uses graphical notations that resemble other modern object-oriented methods there are major differences. SHE integrates object-oriented analysis concepts, concepts of real-time, CCS based communication concepts, and architectural concepts (Figure 1).

The semantics of all concepts are carefully defined and woven into a consistent 'fabric' (formal mathematical semantics). The semantics are the basis for POOSL. The SHE method consists of formal semantics, POOSL and an object-oriented analysis and design method. This latter part defines the preparation of the formalisation using graphical notations mainly. Although the formal semantics are so strongly emphasised in this paper it does not play a role during the practical use of the method. The method is 'designed' to offer concepts that are as natural as possible. POOSL is therefore constrained to a very limited yet expressive collection of primitives.

Objects

SHE distinguishes data objects and process objects. This distinction is also present in the semantics of the language and reveals itself in separate collections of primitives for the behaviour description of process objects and data objects. Both data objects and process objects are defined in classes for the purpose of reuse.

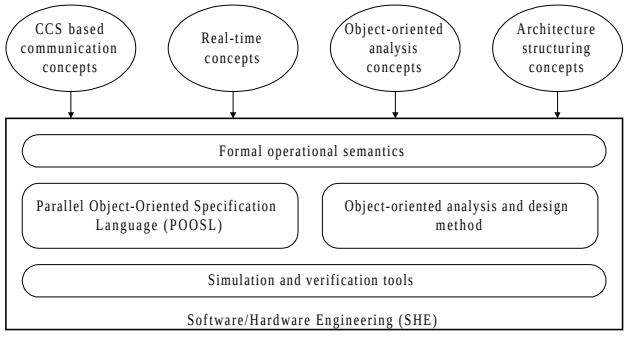

Figure 1. Concepts for SHE method.

Process objects are defined such that they can represent autonomous entities of behaviour. Process objects are asynchronous concurrent entities. They communicate by passing messages. The internal behaviour may consist of several parallel activities. This enables sending and reception of messages simultaneously. Messages are passed via static channels connecting process objects. Behaviour of process objects may be infinite and may be interrupted or aborted.

All data in process objects as well as all data contained in messages are represented by *data objects*. They can be used to construct complex data structures. Operations are performed sequentially. In contrast to process objects, data objects are quite similar to the concept of object in Smalltalk and C++.

Conceptual choices discussed above make POOSL suitable for distributed reactive systems. POOSL handles parallelism and sharing in a well defined way. A data object is never shared by processes objects. Process objects encapsulate data objects. On the other hand data objects can be shared by parallel activities in a process object. This form of sharing is safe because behaviour of data objects is atomic.

CCS-based communication concepts

Initially POOSL was designed with a single CCS based communication primitive (synchronous message passing). Synchronisation and communication of two processes can be described with one such primitive. This gives clear and abstract models that are very suitable for system level design. There is no limitation to construct asynchronous (buffered) communication with the same primitive. Recently POOSL is extended with asynchronous broadcast communication. Interrupt and abort primitives enable interrupt driven communication. Besides communication concepts, CCS also offered basic ideas for parallel composition, selection, channel hiding and channel renaming.

Architecture structuring concepts

Process objects connected by channels offer a basis for the description of a system structure. Processes can be composed to clusters that are interconnected by channels. Clusters are defined in cluster classes. Their behaviour is completely determined by the parallel composition of the process objects and clusters in it. Process objects and clusters are used as abstraction mechanism to model a problem domain. Architecture constraints are denoted on clusters. The properties of subsystems are denoted on the cluster that encompasses it. Examples of such properties are internal parallel behaviour (in contrast to sequential behaviour), physical distribution, and implementation boundaries.

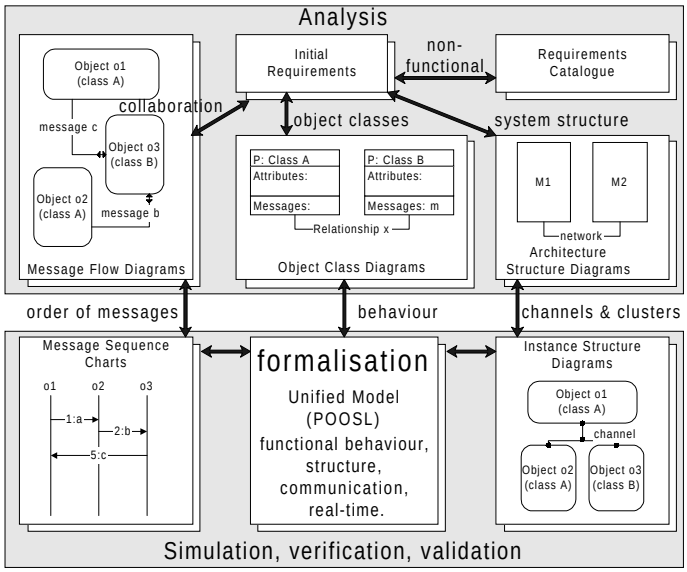

Figure 2. SHE activity framework

Real-time

Among the many requirements and aspects that influence architecture structure there are some that have top priority. Performance requirements and real-time requirements are of this kind. POOSL has an notion of hard real-time, that enables description of time related behaviour and validation of real-time properties [8].

5. Concepts used in practice

Figure 2 shows the SHE method's notations and its flow of activities. Message Flow Diagrams, Object Class Diagrams and Message Sequence Charts are widely used graphical representations in object-oriented analysis methods. Various details of graphical symbols in SHE are matched to the semantics of POOSL. This guarantees that the information gathered during the creation of the diagrams can be formalised. Views on the system, that are offered by various sorts of graphical representations, all have their own value. Each view offers a complementary abstraction of the ' complete' information in the model.

An example of a view is an Object Class Diagram. Such a diagram shows *object classes* and their associations. It is a very abstract view that gives no clue about the number of instantiations of each class, or about the collaboration of objects. Class symbols are designed to match the semantics of language concepts with analysis concepts. The diagrams show Process classes and Data classes. Message names are annotated to SHE class symbols instead of method names (operation names). This is a result of a complete detachment of methods (operations) and messages in process objects. This choice is connected with system level modelling requirements (infinite behaviour, tail recursion, parallelism, etcetera). This is a concrete example of considered coalescing of concepts, which makes the design of a method so complex.

SHE strongly emphasises the search for objects (instances of classes) and their collaborations. Creating a functional system specification includes the search for an appropriate collection of ' things' that conceive the system. These are objects, messages, operations in objects, clusters and communication channels. A system *structure* is explored and prepared using free style pictures called Architecture Structure Diagrams. Physical and/or logical distribution over subsystems as well as an interconnect topology is designed, based on constraints that are available on system level. The results are formalised into Instance Structure Diagrams (Drawing window, Figure 3). They specify a proposed system structure in the form of clusters of process objects connected by channels.

So far the description gave a brief overview of the approach to a system structure and domain exploration. Figure 3 shows a simulation result. Simulation requires formalisation of behaviour of all classes. Edit windows (not shown here) can be opened to enter POOSL class descriptions.

POOSL is an imperative language which is easy to learn because of its simplicity. A behaviour description yields an executable program. It is a considered abstract system level 'implementation.' This enables validation of various system properties before a final detailed implementation in a real system is created. The viability of verification of particular properties is related to the level of abstraction of the model. A final (low level) implementation contains so much detail that it hinders various forms of verification.

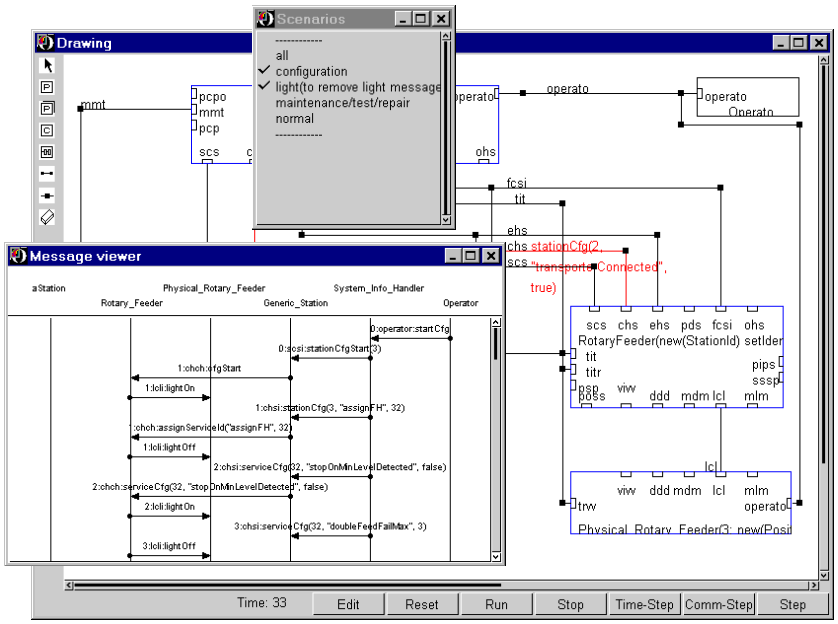

Figure 3. Simulation result of industrial application

Message Flow Diagrams (Figure 2) are used to discuss and prepare formalisation of behaviour of process objects. These diagrams show objects and clusters (rounded rectangles) and the messages they exchange (on arrows). Various sorts of message symbols denote messages with or without reply, interrupting messages, etcetera. Separate Message Flow Diagrams are used to show different scenarios of behaviour. Scenarios enable the presentation of various parts of the system behaviour separate from each other [7]. This is achieved by hiding all channels and objects irrelevant for a particular scenario. This enables a focused discussion with specific experts during the first informal modelling phase. Scenarios define series of causally chained events and operations. Scenarios play a role during informal specification, formalisation, verification, validation and testing. Figure 3 shows a scenario window. Only channels, objects and messages of marked scenarios are visible. Message Sequence Charts are used to specify and view the order of messages in a scenario and to verify the order and time of them during simulation(Message viewer, Figure 3).

The first goal of modelling is to explore system properties in a cost-effective way. Therefore it will be wise to explore performance bottlenecks in separate and dedicated SHE models that can be developed as intermezzos in the overall specification design process.

The final goal of modelling is to give a secure and fast path to implementation (synthesis). So far we realised a prototype translator of POOSL to C++. On the hardware design side we performed an implementation for a processor down to silicon layout [10] using a mapping from POOSL to IDaSS [11] as intermediate to VHDL.

Figure 3 is borrowed from an industrial research project with Buhrs Zaandam b.v. and TNO-TPD (Delft). The project aims at investigating the use of system level design for the development of complex industrial controllers. Although our main interest is in the joined area of telecommunication and information processing systems, we accepted this project because it confronts with a whole variety of challenges of specifying and designing a complex distributed reactive hard real-time system. This project is in the implementation phase now. It gave us valuable feedback for the development of our concepts and tools.

6. Conclusions and future research

This paper reports on a concrete method for system level design and describes an approach towards the challenges of system level design methodology research. This section gives a summary of necessary research (Figure 4).

The concrete SHE method aims at object-oriented specification of system behaviour, structure and real-time properties. Method and tools based on formal semantics have been developed. These tools include a compiler, a simulator and translators to Smalltalk and C++. This research directed to concepts of real-time, parallelism, typing and object-orientation will be continued. Besides research on system level analysis we will extend research on formal verification. A tool is available to verify system communication based on CCS-oriented abstractions. This path is relatively easy as result of the considered conceptual choices in the design of POOSL. However for

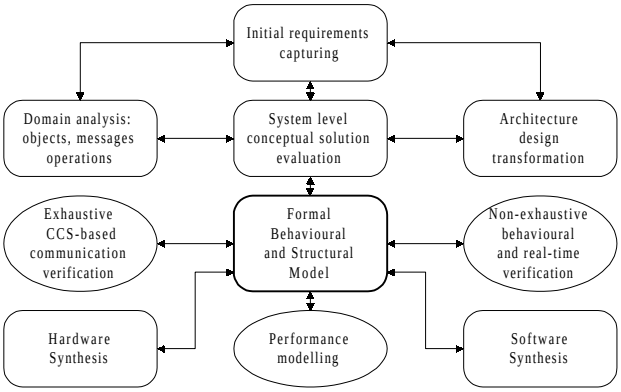

Figure 4. Selected research

the time being, simulation is the main tool to explore the functional behaviour of a model [9].

Because of the fundamental problems with exhaustive verification a collection of various approaches and tools will be necessary. Where applicable, exhaustive verification must be supported. Elsewhere non-exhaustive verification should be applied. It will be very hard to produce portable tools unless formally based methods are developed. Various forms of verification will require very particular abstractions of complete models. In general verification will require a collection of dedicated models of the same system. They can only be derived from a model if its formalisms are designed to support well-defined strategies for system level design and verification.

Verification and validation requires adequate languages expressing system properties and parameters for quality metrics, such as real-time temporal properties, assertions, invariants, and performance parameters. Much research will be necessary to bridge the gap between requirements capture and formalism. Systematic development of formal models and systematic deduction of formal properties and quality parameters should be addressed. Besides formal (non-exhaustive) verification and testing of models against properties, the estimation or calculation of quality parameters should be addressed.

System level architecture design is supported by research results on behaviour-preserving structure transformations [9]. Hardware synthesis will be explored by mapping system level models onto hardware description languages. Software synthesis will be explored by mapping system level models onto programming languages. The use of explicit real-time operating systems to implement concurrency will be explored besides solutions without the use of explicit real-time operating systems. Research results will be validated mainly in the application area of VLSI design and tele/data-communication.

7. REFERENCES

[1] D. Gajski. Methodology is the future. In proceedings of APCCAS'96, IEEE, New York, 1996, pp. 269-277.

[2] P.H.A. van der Putten; J.P.M. Voeten. Specification of reactive Hardware/Software Systems, Ph.D. thesis, Department of Electrical Engineering, Eindhoven University of technology, Eindhoven, The Netherlands, 1997.

[3] J.M. Daveau; G. Fernandes Marchioro; C.A. Valderrama; A.A. Jerraya. VHDL generation from SDL specifications, IFIP TC10 WG10.5 Int. Conf. on Computer Hardware Description Languages. and their Applications, Chapman & Hall, London, 1997, pp. 182-201.

[4] J. Leao da Silva; C. Ykman-Couvreur; G. de Jong; B. Lin; H. De Man. A system design methodology for telecommunication network, Proceedings Great Lakes Symposium on VLSI '96, IEEE Comput. Soc. Press, Los Alamitos, California, 1997, pp. 64-69.

[5] B. Selic, G. Gullekson, and P.T. Ward. Real-Time Object-Oriented Modeling. Wiley & Sons, New York, 1994.

[6] UML. Rational Software Corporation, 1997, Available via www: http://www.rational.com/uml

[7] P.H.A. van der Putten; J.P.M. Voeten; M.C.W. Geilen.; M.P.J. Stevens. Multidisciplinary scenarios in Software/Hardware Engineering, in Proceedings of ProRISC/IEEE workshop, J.P. Veen, Ed., STW Technology Foundation, Utrecht, The Netherlands, 1997, pp. 461-468.

[8] M.C.W. Geilen; J.P.M. Voeten. Real-Time Concepts for a Formal Specification Language for Software/Hardware Systems, in Proceedings of ProRISC/IEEE'97, J.P. Veen, Ed., STW, Technology Foundation, Utrecht, The Netherlands, 1997, pp. 185-192.

[9] J.P.M. Voeten, P.H.A. van der Putten, and M.P.J. Stevens. Behaviour-Preserving Transformations in SHE: A Formal Approach to Architecture Design. in Proceedings of EUROMICRO'96, P. Milligan and K. Kuchcinski, ed., IEEE Computer Society Press, Los Alamitos, California, 1996, pp. 19—27.

[10] R. Michielsen and J. Voeten, Implementation of POOSL in Hardware, in Proceedings of ProRISC/IEEE'97. J.P. Veen, Ed., STW, Technology Foundation, Utrecht, The Netherlands, 1997, pp. 427-434.

[11] A.C. Verschueren. An Object-Oriented Design and Simulation System for VLSI, in Microprocessing and Microprogramming, Vol. 30, Numbers 1-5, August 1990, pp. 241-246.