Embedded UML: a merger of real-time UML and co-design

Grant Martin
Cadence Design Systems, Inc.
555 River Oaks Parkway
San Jose CA 95134 USA
+1-408-894-2986
gmartin@cadence.com

Luciano Lavagno
Cadence Design Systems, Inc.
2001 Addison Street, Third Floor
Berkeley, CA 94074 USA
+1-510-647-2810
luciano@cadence.com

Jean Louis-Guerin
Cadence Design Systems, Inc.
18 Rue Grange Dame Rose
78148 Velizy France
+33-134-885-317
jlg@cadence.com

ABSTRACT

In this paper, we present a proposal for a UML profile called 'Embedded UML'. Embedded UML represents a synthesis of various ideas in the real-time UML community, and concepts drawn from the Hardware-Software co-design field. Embedded UML first selects from among the competing real-time UML proposals, the set of ideas which best allow specification and analysis of mixed HW-SW systems. It then adds the necessary concept of underlying deployment architecture that UML currently lacks in complete form, using the notion of an embedded HW-SW 'platform'. It supplements this with the concept of a 'mapping', which is a platform-dependent refinement mechanism that allows efficient generation of an optimised implementation of the executable specification in both HW and SW. Finally, it provides an approach which supports the development of automated analysis, simulation, synthesis and code generation tool capabilities which can be provided for design usage even while the embedded UML standardisation process takes place.

Keywords

UML, embedded systems, real-time systems, HW-SW co-design, function-architecture co-design, platforms.

1. INTRODUCTION

The last several years of development have seen the emergence in more pragmatic form of hardware-software co-design tools among the more 'hardware-centric' EDA community, for example, Cadence's Virtual Component Co-design tool VCC [5], Synopsys's CoCentric tool [31], and CoWare N2C. The software community, after several years of work, converged on a set of notations for developing specifications of object-oriented systems known as the Unified Modelling Language or UML [26]. However, until recently, these worlds have remained separate.

The rise of embedded real-time systems, consisting of significant amounts of variable hardware and software, and the introduction of deep submicron IC processes, allowing the emergence of complete Systems-On-Chip (or chipsets), force a unification of these separate worlds. Functionality must be specified independent of implementation. It must be possible to specify flexible hardware-software architectures on which that function will be implemented. Commitment to particular choices of components should be delayed as much as possible to the end of the design process. Exploration of various architectures and component choices should be encouraged with a variety of static and dynamic analysis techniques.

The general notion of 'function-architecture' co-design [2] has been introduced within the hardware-software co-design community and has now been widely adopted. The concept consists of an orthogonal capture of undifferentiated system functionality in the form of an executable specification model; the definition of a basic HW-SW architecture on which that system is to be realised; and the construction of an explicit mapping between the two views which provides the essential hardware-software partitioning and assignment of functions to components. This then allows both formal and informal analysis and simulation of the design to proceed, serving as the basis for design space exploration.

The notion of flexible hardware-software architectures has been refined into the concept of a 'platform' [11] and platform-based design [6]. The concepts of platform and function-architecture co-design can be fruitfully married together [18]. A platform, from the viewpoint of the systems designer, can be considered as an 'Application Programmer's Interface' (API) view of the set of resources and services offered by the platform to the system implementers. This abstraction level or API allows the user to configure the target system platform to best support the application, within the limits of system configurability. It also allows the application mapping to system resources to be optimised either by hand or through automation-assisted means.

In the UML community, where standardisation occurs via processes managed by the Object Management Group (OMG) [20], many proposals have arisen for extensions to UML that better support the development of real-time systems. In the OMG these are called UML 'profiles': collections of specific extensions to UML notations (called 'stereotypes', tagged values and constraints) and associated semantic notions, bundled together in a collection of concepts which are suitable for expressing in an effective manner the concerns of a particular design domain.

In the next section, we will review the UML, and then discuss a number of the proposals made by various groups for real-time UML profiles, including the OMG UML revision task force.

CODES'01, April 25-27, 2001, Copenhagen, Denmark.

2. THE UML

The UML is a collection of notations [4, 12] for capturing a specification of a software system. These notations have a formal syntax defined by the OMG but often only informal semantics. They are primarily graphical, with textual annotation. Certain notations have particular importance for modelling embedded systems; these include:

- Class diagrams show the static structure of the specified system, in particular, classes, their internal structure including attributes, their methods, and their relationships to other classes (such as inheritance or generalisation, and associations)
- Collaboration diagrams define a specific way to use objects in a system by showing the possible interactions between them
- Sequence diagrams (very similar to SDL's Message Sequence Charts or MSC's) are another form of interaction diagram. Sequence diagrams in particular could be used to help create testbenches.
- Use case diagrams are a way of associating the 'actors' and 'use-cases' in a system via the use of relationships. This has potential again in helping to create verification scenarios/testbenches.
- State diagrams are based on Harel statecharts [15]. UML state diagrams can model object behaviour over several use cases.
- Deployment diagrams are a rather weak attempt at showing the run-time configuration of software and hardware components in a system. Deployment diagrams are often combined with Component diagrams, which attempt to depict packages of object implementations and their interfaces. A much stronger notion of 'mapping' between a logical model (functional blocks) and an engineering model (a platform, with its architectural resources), and a concept of modular resource models (also known as architectural services) in API form, are required for effective embedded system development.

We can summarise the usefulness of the UML standard as essentially a set of notations for specifying the requirements, documenting the structure, decomposing into objects, and defining relationships between objects, in a software system. By adopting UML notations, development teams can communicate among themselves and with others using a defined standard. This usage seems most common among early adopters of UML.

Adopters of particular tools can also have to some extent, executable specifications and code generation via class, state and sequence diagrams. The lack of formalised semantics makes these capabilities tool-dependent and proprietary, with no guarantees of interoperability; in fact, the stereotypes proposed are in general, mutually incompatible. Furthermore, it complicates the communication of UML models between users. This is a key lack of UML, which is being studied by a number of OMG committees.

3. REALTIME UML EXTENSION PROPOSALS

The deficiencies of standard UML as a vehicle for complete specification and implementation of real-time embedded systems has led to a variety of competing and complementary proposals. Some of these have been in response to OMG RFI's and RFP's; others have been proposed based on specific tool supported methodologies. A good summary of some of the debates and issues is found in [14]. We can classify the various proposals into the following areas:

- Use standard UML with stereotypes for real-time support
- Functional decomposition, or object structuring, imposed on top of UML's object-oriented notations
- Translation from UML into other languages for real-time implementation
- Develop well defined Action Semantics for UML
- Introduce the notions of hardware and software architecture modelling, and a logical to engineering model 'mapping'
- Introduce the notions of schedulability, performance and time into UML via a profile

3.1 Use UML as is, or simplify it

This concept has been promoted by Bruce Powel Douglass, most notably in [8]; his basic philosophy is stated very clearly: "It is important to note that the UML is adequate for the complete development of real-time embedded systems." [8, p. 52]. Some people have proposed simplifying UML – for example, state diagrams, as in [19]. While Douglass uses UML as is for real-time, he does take advantage of its stereotyping extension capability, using basic UML notations, to "help explain the purpose of model elements that occur commonly within the real-time and embedded domains". These include:

- UML active objects as the roots of OS threads
- A variety of UML messages to represent synchronisation and message arrival patterns
- A guarded operation stereotype to model semaphores
- Node stereotypes to represent HW architectural elements: processors, devices, sensors, actuators, displays, etc.
- Component stereotypes for SW components

3.2 Functional Decomposition, or Object Structuring of System

We know of two major attempts to define 'object structuring' of systems in UML – what might also be called 'functional decomposition'. These are the UML-RT profile, and the SDL-UML profile, both based on existing languages (ROOM and SDL). The UML-RT profile derives from the ObjecTime ROOM methodology [27], which is now available as the Rose-Real Time tool from Rational. It has been proposed as an extension of UML for real-time in an influential white paper by Selic and Rumbaugh [28] and an article [29]. In fact, so influential have these papers been that many people believe that Real-time UML has been standardised around the notions in these books and papers. In reality, these notions are only in the RFP stage for UML 2.0, albeit described rather cryptically as "support encapsulation and scalability in behavioural modelling, in particular, for state machines and interactions", and as architectural modelling constructs. [21]

The key concepts in functional decomposition as defined in ROOM etc. are:

- Capsules are used to represent the major behavioural objects of a real-time system. These may be functional computation units, in many senses quite orthogonal to object oriented decompositions. In ROOM, capsules incorporate state machine functions, and can be hierarchical. They have explicit external interfaces.
- Explicit communications structures: ports, protocols and connectors. This provides a single queued asynchronous communications mechanism. These are used in capsule collaboration diagrams that graphically depict the functional decomposition, or 'object structure' of a system
- ROOM and RoseRT also provide an underlying virtual machine model and interface to runtime RTOS services such as messaging, as a way of keeping the functional specification layered on top of commercial processor/RTOS ports of the tool, and supporting target-specific code generation.

The SDL-UML profile proposal is described next.

3.3 Translation from UML into other modelling notations for implementation

A notable advocate of this approach is Telelogic in their Telelogic Tau toolsuite. The UML suite allows a user to capture specifications in UML (class diagrams, state diagrams and sequence diagrams), and then via a UML to SDL translation, move to an SDL based toolsuite for real-time code generation (in C, C++ and other languages). The SDL suite also offers functional simulation and test generation (TTCN). In this case, only those UML notations that have 'natural' analogues in the SDL world are translatable. The concept, called the SOMT method (SDL-oriented Object Modelling Technique) was described several years ago by Ek [10]. More recently, the objective of this approach was described succinctly as to "permit the expressive power of UML to coalesce with SDL's strengths of coherence and semantics" [3]. Recent work by the ITU-T has led to a specification proposal Z.109 for SDL combined with UML that defines the mapping of translatable concepts between the notations.

3.4 Action Semantics for UML

This proposal is somewhat orthogonal to others, in that it applies to all UML development, whether for real-time embedded systems or any other software application. UML has been defined with a formal syntax, but not formal semantics. As the OMG RFP for action semantics stated: "the UML currently uses uninterpreted strings to capture much of the description of the behaviour of actions and operations. To provide for sharing of semantics of action and operation behaviour between UML modellers and UML tools, there needs to be a way to define this behaviour in a well-defined, interoperable form. At such time as the Action Semantics requested in this proposal are mapped to a syntax, and are combined with the UML, the UML shall constitute a computationally complete language. This language is targeted at system analysis and behaviour description, and is not envisioned to be language suitable for system deployment." [22]

Clearly, the lack of defined action semantics, among many other deficiencies, has kept UML in the documentation domain and prevented its use for interoperable executable specifications. An extensive response (200 pages) to the RFP has been submitted [24]. This is not a new programming language – "Rather, the action semantics provides for the specification of systems in sufficient detail that they can be executed, and the actions semantics should provide just enough semantics to enable the specification of computation". These semantics may be standardised in UML 1.4, 1.5 or a later version.

3.5 Architectural Modelling and Mapping

The notion of a logical architecture, a physical or engineering architecture, and a 'mapping' between them, show up in several places in UML real-time extension proposals, as well, of course, in the function-architecture co-design concepts discussed earlier. Of course, the original UML specification had the weak notions of deployment diagrams.

Artisan Software in their Realtime Studio, for example, has added the notions of system and architecture modelling, a system architecture diagram, and mapping of model artefacts (such as real-time tasks) to elements (resources) on the architecture diagram. [1]. The OMG RFP for a UML profile for Scheduling, Performance and Time [23] asked for models of resources, physical (HW) and non-physical (SW, e.g. semaphores, queues), and deployment of software components to physical resources. The joint multi-company submission in response to this RFP includes logical (application or client) and engineering (platform) models, representing functions and architectures, and a relationship between elements in each called a realisation relationship. [25] In addition, resources have services that they offer to clients, which have a strong relationship to the co-design notion (in VCC) of an 'architectural service'. [32]. These concepts are briefly summarised in [30].

3.6 Schedulability, Performance and Time

The OMG RFP for UML profile for Schedulability, Performance and Time also deals with the notions of modelling time and clocks, concurrency, and support for analysis of performance and schedulability, in addition to resource modelling. This includes scheduling policies, timing specifications and constraints, timing services, and visual representations of these notions linked to behaviour.

The response to the RFP discussed above has extensive definitions of class-based stereotypes based on the notion of Quality of Service (QoS) characteristics for resources. QoS characteristics will serve as the base for real-time embedded systems engineering [30]; the relationship between application objects and resources will involve 'required QoS' (the constraint) and the 'offered QoS'. Underlying the QoS notions is an extensive set of classes for time modelling, real-time stereotypes, and a schedulability model [25].

4. EMBEDDED UML: THE PROPOSAL

So what is to be done? "The Philosophers have only interpreted the world in various ways; the point, is to change it."

Embedded UML is a research project with the goal of defining, and proposing, a UML profile suitable for embedded real-time system specification, design and verification. It represents a synthesis of the best concepts in UML, real-time UML, and function-architecture co-design, married to the concept of platform-based design.

Embedded UML retains the best of UML and real-time UML:

- The specification of embedded systems as a collection of re-usable communicating blocks using a functional decomposition.
- Class diagrams for object definition
- Encapsulation of functions within a 'block', an extension of a capsule
- Communications explicitly defined via ports, protocols and connectors
- Use cases and sequence diagrams to specify testbenches and test scenarios
- Carefully defined state diagram semantics, combined with specified action semantics which can be used to drive code generation, optimisation and synthesis.
- The concept of a refinement continuum, from non-executable specifications, through formal executable specifications, through to implementation.

Then, complementing these concepts, we add from the co-design world what is missing:

- A rigorously defined platform model in both HW and SW for the implementation architecture. Conceptually, this can be thought of as a collection of resources offering services, as in UML real-time extension profiles. The collection of platform services can be thought of as a system platform 'API'.
- Using 'mapping' as the platform-dependent refinement paradigm for performance analysis, communication synthesis and optimised code synthesis/generation.
- A concept of 'reactive' rather than UML's 'active' objects, for blocks.

4.1 Reactive Objects

Embedded UML relies on reactive, not active objects [16]. An active object in UML is defined as "an object that owns a thread and can initiate control activity". Each active class owns a single thread, and an event queue. It executes a never-ending event-loop that takes events from the queue and injects them to the target objects. This is one specific model of computation; to better model embedded systems, we need multiple models of computation with flexible communications.

Reactive objects are a better match to embedded systems consisting of concurrent processes mapped to multiple hardware resources with asynchronous communications between themselves and 'reacting' to external events and stimuli. A reactive class is one that can react to events; i.e., an event consumer. Reactive objects must specify a control structure for the object, in the form of state diagrams or code (following formal action semantics), a communication structure between objects with rigorous interfaces (via ports and connectors, similar to a capsule collaboration diagram). Communications and synchronisation mechanisms cannot be fixed, but must support a variety of synchronous and asynchronous styles.

4.2 Communications Mechanisms

Embedded UML must support multiple means of communications. It is important to allow customisation of the communications protocol at the specification level of abstraction, and its detailed implementation. Communications abstractions must include point-to-point mechanisms, task input queues, abstract messaging, discrete event; and a variety of implementation styles must be able to be modelled – e.g. finite buffers, interrupts, semaphores, shared memory, etc. This is because different application domains and different implementation platforms have different requirements. For example, multimedia applications can be modelled naturally as an interconnection of blocks (performing operations such as filtering, decoding and so on) interconnected by FIFO queues. Telecommunications protocols, on the other hand, are better suited for prioritised queues for message handling. Automotive applications require non-queued events whose priority is determined dynamically by the receiving object, and so on.

Communication refinement must allow modelling to incrementally define an elaboration methodology from abstract events (with undefined protocols), through specific mechanisms (e.g. infinite event queues or lossy buffers), through specific implementations (for example, SW driver through a bus protocol to a multi-write HW register). These implementations of communications need to be able to model the implications on Quality of Service (QoS) of shared resources – e.g. a multi-mastered, arbitrated bus. As an example, consider the refinement of a FIFO queue that was originally specified as unbounded, between abstract reactive objects, without a detailed implementation choice. The first step is the choice of an implementation decision for the objects. If both are implemented as software on the same processor, then a static or quasi-static scheduling algorithm can be used to merge both into a single task, and a shared circular buffer is the best implementation option for the FIFO. If one is implemented in hardware and one in software, then the FIFO communication must be implemented using the available communication resources: interrupts, DMA, shared memory and so on. Clearly, a detailed specification of this implementation in early stages of the design is an over-specification. Embedded UML supports the user also in this refinement and implementation process.

Communications refinement involves both the communications mechanisms and the scheduling mechanisms that control access to the resource-bound services. We propose a communication services stereotype in Embedded UML that is used for implementing interfaces for communications refinement (as in VCC), and provides via libraries common mechanisms used in abstract specifications, such as dataflow FIFOs, SDL-type queues, events, etc. Interface refinement involves substitution of detailed models while preserving properties, as described for example in [13]. In this framework, each component of the architectural model (also known as engineering model or platform) implements services for users (other architectural components or functional components). For example, the CPU architectural component offers to both user code (functional

component) and the RTOS (another architectural component, that is part of the engineering model) a data read and write service, as well as an instruction fetch service. These services can be invoked by the upper layers whenever they need to model the performance of instruction fetches and data loads and stores. Such upper layers need not know whether the CPU has a cache or not, whether the bus supports burst access or not, or the bus frequency. Information hiding and modularity are essential elements of a re-usable architectural modelling strategy. The CPU component will in turn use cache services and bus services to model these aspects.

A key element of an effective modelling strategy is allowing the architectural designer to quickly change the configuration, and thus the performance model, to reflect architectural changes. In the mapping mechanism, the binding between service use and implementation (determining, e.g., whether or not a cache is used) depends on the structure of the architectural model. The architectural netlist becomes the guide to binding the appropriate uses to the appropriate declarations.

Synchronisation will also require the definition of a synchronisation services stereotype used to support refinement of scheduling requirements into implementations. For example, the designer may want to specify that an input sampling object must be executed once every millisecond, with a maximum jitter of 10%, or that two objects have a relative execution priority.

4.3 A Path to Implementation

Today's state of the art for code generation involves either target-independent automated code generation (combined with mapping libraries that allow correct execution on particular runtime processor/RTOS platforms), or user-defined manual optimisation for a particular implementation architecture. In the first case, the code is inefficient, possibly in both memory footprint and execution performance; in the second case, the process is slow, painful and requires considerable platform, application and software generation expertise.

We believe that Embedded UML needs to offer a new approach. The designer will map a functional, executable specification to a platform architecture. This is used to derive a performance analysis model for simulation and static analysis. In addition, the mapping of function to architecture will be used as the basis for automated derivation and generation of optimised target-specific implementations of function (SW on DSPs, SW on general processors, HW, …), communications mechanisms (SW-HW, SW-SW, HW-HW) and synchronisation (static scheduling, interrupts, pre-emption policies, etc.) [2, 7, 9, 17].

Mapping as an implementation paradigm is the fundamental base for target-dependent code synthesis and optimisation. Mapping will be used in synthesising functional blocks, with target-specific code generators and RTOS customisation for task creation, priorities and scheduling. It will also control communications synthesis: using available platform resources (memories, buses) and appropriate refinement patterns or templates (interrupts, polling, DMA, etc.). This will not, initially, be totally automated; user intervention to create the mapping and set parameters will be required. However, dramatic improvements in generated code efficiency and footprint should be possible over target-independent automated methods. Fewer manual changes for optimisation will be required *post facto*.

4.4 Summary of Embedded UML

In summary, Embedded UML will use blocks, an extension of the capsule notation, for functional encapsulation. Netlists, an extension of collaboration diagrams, will be used for functional composition. Interfaces and channels, extensions of ports and signals, will be used for communication specification and refinement, along with stereotypes for communication and synchronisation services. Finally, mapping diagrams, a rigorous extension of deployment diagrams, will be used for performance analysis and optimised implementation generation.

5. EMBEDDED UML AND CODESIGN

We use VCC as an example of a function-architecture co-design tool and methodology that can serve as a suitable base for the development and elaboration of Embedded UML. VCC 2.0 is a partial superset of the embedded UML profile, offering customisable communication refinement (via architectural services modelling), architectural diagrams covering HW and SW resources, mapping diagrams, performance analysis and links to implementation (HW and SW) flows.

In this respect VCC has similar characteristics to other co-design concepts and tools in the literature, although in an advanced, commercially available form. It is particularly strong in architectural modelling, mapping, and performance analysis, along with rich modelling capabilities for communications refinement. Lacks in VCC and in other co-design concepts include (currently) no equivalent to UML class diagrams (for object-oriented model creation), class inheritance, sequence diagrams (for testbench creation), use cases (for scenario elaboration), and customisable abstract models of communication. However, these kinds of capabilities could be added to VCC and other co-design tools to support the development of Embedded UML and its demonstration in real product design flows.

6. CONCLUSIONS

We have analysed the deficiencies of UML to support the specification, development and implementation of embedded real-time software-based systems running on malleable HW-SW platforms. We have then surveyed the wide variety of proposals available within the real-time UML community to overcome these lacks, and have indicated some key capabilities available in HW-SW co-design research and tools which can provide support for such systems. Finally, we have created a Hegelian synthesis of the best UML/real-time UML and co-design concepts, to propose a new merged Embedded UML profile.

The standardisation of something like Embedded UML, and its adoption in embedded software, system and co-design tools over the next few years will have a radical effect on system and software design and implementation productivity. This paper is part of a rallying cry for the disparate EDA/co-design and SW/UML communities to unite into bringing this vision into reality. When worlds collide, a new world can come into being.

Future areas of research that are needed to move the vision along will include: requirements specification, recording,

decomposition and tracking from product definition through to implementation; specific techniques for code optimisation for a platform, especially in targeting specific compiler and processor capabilities, memory hierarchy management, and efficient use of RTOS APIs; and the dual problem of optimising the platform for particular applications domains.

7. ACKNOWLEDGMENTS

Our thanks to all colleagues with whom we have discussed these ideas; particular thanks to Ellen Sentovich and Ian Dennison.

8. REFERENCES

[1] Artisan Software, "Real-Time Studio: The Rational Alternative", white paper, version 3.0, 29 July 1999, URL: http://www.artisansw.com/whitepapers.

[2] Balarin, F., Chiodo, M., P. Giusto, Hsieh, H., Jurecska, A., Lavagno, L., Passerone, C., Sangiovanni-Vincentelli, A., Sentovich, E., Suzuki, K., and Tabbara, B. Hardware-Software Co-Design of Embedded Systems: The POLIS Approach, Kluwer, 1997.

[3] Bikander, M. "Graphical Programming Using UML and SDL", IEEE Computer, December 2000, pp. 30-35.

[4] Booch, G., Rumbaugh, J. and Jacobson, I., The Unified Modeling Language User Guide, Addison-Wesley, 1999.

[5] Chakravarty, S. and Martin, G. A new embedded system design flow based on IP integration, in Proceedings of DATE 99 User's Forum , Munich, 1999, 99-106.

[6] Chang, H., Cooke, L., Hunt, M., McNelly, A., Martin, G. and Todd, L., Surviving the SOC Revolution: A Guide to Platform-Based Design, Kluwer, 1999.

[7] Cortadella, J., Kondratytev, A., Lavagno, L., Massot, M., Moral, S., Passerone, C., Watanabe, Y., and Sangiovanni-Vincentelli, A., "Task Generation and Compile-Time Scheduling for Mixed Data-Control Embedded Software", Design Automation Conference, June 2000, pp. 489-494.

[8] Douglass, B. P. Doing Hard Time – Developing Real-Time Systems with UML, Objects, Frameworks and Patterns, Addison-Wesley, 1999.

[9] Edwards, S. "Compiling Esterel into sequential code", Design Automation Conference, June 2000, pp. 322-327.

[10] Ek, A. "The SOMT Method", Telelogic white paper, September 19, 1995.

[11] Ferrari, A. and Sangiovanni-Vincentelli, A., "System Design – Traditional Concepts and New Paradigms", Proceedings of ICCD 99, Austin, October, 1999, pp. 2-12.

[12] Fowler, M., with Scott, K., UML Distilled, Second Edition, Addison-Wesley, 2000.

[13] Gajski, D., Zhu, J., Dömer,, R., Gerstlauer, A. and Zhao, S., SpecC: Specification Language and Methodology, Kluwer Academic Publishers, 2000.

[14] Ganssle, J. "Navigating through new development environments", Embedded Systems Programming, May, 1999, pp. 22-30.

[15] Harel, D. "Statecharts: A visual formalism for complex systems", Science of Computer Programming, Vol. 8, 1987.

[16] Lavender, R. and Schmidt, D. "Active Object: An Object Behavioral Pattern for Concurrent Programming", in Pattern Languages of Program Design 2, eds. Vlissides, J., Coplien, D. and Kerth, M., Addison-Wesley, 1996.

[17] Lee, E. and Messerschmitt, D., "Static Scheduling of Synchronous Data Flow Graphs for Digital Signal Processing", IEEE Transactions on Computers, Jan. 1987.

[18] Martin, G. "Productivity in VC Reuse: Linking SOC platforms to abstract systems design methodology", Forum on Design Languages: Virtual Components Design and Reuse, Lyon, September 1999, pp. 313-322.

[19] Mellor, S. and Selic, B., with Barr, M. "A UML Point/Counterpoint" – "Modeling Complex Behavior Simply" and "How to Simplify Complexity", Embedded Systems Programming, March 2000, pp. 37-56.

[20] OMG web site URL: http://www.omg.org/

[21] OMG Request for Proposal: UML 2.0 Superstructure RFP, OMG document: ad/2000-08-09.

[22] OMG Request for Proposal: Action Semantics for the UML RFP, OMG document: ad/98-11-01.

[23] OMG Request for Proposal: UML Profile for Scheduling, Performance and Time, OMG document: ad/99-03-13.

[24] OMG Response to OMG RFP ad/98-11-01, Action Semantics for the UML, Revised September 5, 2000. OMG document: ad/00-09-08.

[25] OMG Response to the OMG RFP for Schedulability, Performance and Time, Version 1.0, August 14, 2000, OMG document: ad/2000-08-04.

[26] Rumbaugh, J., Jacobson, I., and Booch, G., The Unified Modeling Language Reference Manual, Addison-Wesley, 1998.

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

[28] Selic, B. and Rumbaugh, J. "Using UML for Modeling Complex Real-Time Systems", white paper, Rational (ObjecTime), March 11, 1998.

[29] Selic, B. "Turning clockwise: using UML in the real-time domain," Comms. of the ACM, Oct., 1999, pp. 46-54.

[30] Selic, B., "A generic framework for modeling resources with UML", IEEE Computer, June 2000, pp. 64-69.

[31] Synopsy CoCentric SystemStudio can be found at the URL: http://www.synopsys.com/products/cocentric_studio/cocentric_studio.html

[32] VCC information can be found at the URL: http://www.cadence.com/eda_solutions/hcd_l3_index.htm.