A System Design Methodology for Software/Hardware Co-Development of Telecommunication Network Applications

Bill Lin

IMEC, Kapeldreef 75, B-3001 Leuven, Belgium
E-mail: billlin@imec.be
Tel: +32/16/28.15.41 ; Fax: +32/16/28.15.15

Abstract

In this paper, we describe a system design methodology for the concurrent development of hybrid software/hardware systems for telecom network applications. This methodology is based on the results of an investigation and evaluation of an actual industrial system design application for ATM-based broadband networks. The aim of this methodology is to provide an integrated design flow from system specification to implementation.

1 Introduction

Modern telecommunication systems are rapidly increasing in design complexity. Elaborate network management is needed to support a wide variety of broadband multimedia services. The design of these systems is often a collaborative effort taking place at multiple sites. Such complex systems require a combination of both *hardware* and *software* components in order to deliver the required functionalities at the desired level of processing performance and programmability. These hardware and software components must be designed concurrently to minimize time to market.

Example telecom network applications include system components for ATM-based broadband networks, mobile network infrastructures to support GSM-based cellular communication, SONET and SDH based networks, and interactive video-on-demand servers.

In current industrial design practice, specification formalisms are mainly used to model each hardware or software component separately. *System design*, preceding hardware/software partitioning, proceeds still to a large degree in an informal manner. The system specification handed over to hardware and software designers is usually an informal description in the form of a report document, and is often susceptible to ambiguous interpretations. Significant time is spent in interpreting and understanding these specifications, and still a lot of mismatches occur between the expected and the actual behavior of designed components, leading to a lengthy system integration and test phase after component design. Currently, the system integration and test phase can account for nearly 50% of the design cycle for typical telecom network designs in system houses. There is a clear need for suitable system level specification formalisms and validation techniques to overcome this problem.

Once the hardware and software parts are identified, more formal specification models and design techniques are employed. In the case of software, C or C++ is used. In the case of hardware, a register transfer language like VHDL or Verilog is used. The different hardware and software components of a complete system are specified and synthesized separately, which often introduces implementation mismatches, in addition to specification mismatches. Small changes at the system level often require very substantial changes to the hardware or software models of the components. While automated code generators exist for producing machine code from C or C++, and automated hardware synthesis tools exist for producing hardware implementations from VHDL or Verilog, there is a general lack of tools to *bridge the gap* from the global system level design specification to these traditional hardware and software design methods. Moreover, there is a general lack of a coherent design methodology for structuring the system design flow.

In this paper, we outline a design methodology and a system design flow for the design of hybrid software/hardware systems that are typically found in telecom network applications. This methodology is based on the results of an investigation and evaluation of an actual industrial system application for ATM (Asynchronous Transfer Mode) based broadband networks at Alcatel Bell [26]. Specifically, a user transparent connectionless router for interconnecting geographically distributed high speed local area networks to an ATM connection oriented public transport network has been investigated. This case study is described in Section 2.

As a result of this investigation, we have developed a system design methodology based on a concurrent object-oriented programming model as the system behavioral specification formalism. This proposed specification model is described in Section 3. Simulation and debugging tools are provided, leveraging on the wide corpus of existing software development tools, for the early conceptual validation of the system specification before proceeding to the implementation design flow.

For implementation, we provide automated design tools for transforming the system level model into the traditional levels of design entries for hardware and software implementation. In particular, conventional C++ [24] and VHDL [16] models are generated for the parts of the system to be implemented in software and hardware, respectively. New system-level synthesis functionalities are required to achieve these automations, These new functionalities are part of a system design flow, which is outlined in Section 4. It is important to note that traditional hardware and software design methods, mostly commercially available, are used in our design flow, thereby not reinventing new solutions where adequate ones already exist. The remaining two sections of this paper, Section 5 and Section 6, present related work and concluding remarks, respectively. The work outlined in this paper is embodied in a system design compiler called *Matisse*. This system design compiler is incorporated into the *CoWare* heterogeneous design environment under development at IMEC [8].

2 Case Study: Connectionless Router for ATM

As a case study, we have investigated an actual industrial ATM-based broadband network application developed by Alcatel Bell. The application is a user transparent connectionless router called the ACTS (Alcatel Connectionless Transport Server) [26] that provides the necessary functions for the direct provision and support of data communication between geographically distributed computers or local area networks (LANs) over a broadband ATM-cell based transport network.

ATM [9] is a fast packet switching transfer mode that supports high-speed integrated services by splitting all communications into equal 53-byte cells. These cells can be used to carry every kind of information, be it computer data, video, or voice. By using small cells to transfer data, the technology enables networks to provide the flexible multiplexing needed to support a wide variety of traffic, ranging from high to low bandwidths, and from bursty to steady bit rates. In addition, ATM is characterized by a connection oriented mode of operation.

Since local area networks are connectionless oriented, the ATM network needs to be augmented with special user transparent connectionless routers to provide connectionless services. This is the role of the ACTS. This router can for example support Switched Multi-megabit Data Services (SMDS) [2] in a B-ISDN environment. The SMDS protocol features packets of variable length, which are transported by one or more fixed-size ATM cells. The communication service offered is connectionless, meaning that a sender does not have to set up a connection prior to sending data, but can start transmitting data immediately. The router is responsible for storing incoming data, finding out where to route them, and forward them. This situation is depicted in Figure 1.

In its current implementation, the ACTS consists of several custom ASICs and a programmable processor for executive control. In future implementations of similar systems, the functionalities of several of these processors will be integrated into a single VLSI chip, and some functionalities previously implemented in hardware will migrate to software to exploit the increasing processing performance of emerging embedded microprocessors.

3 The Programming Model

In this section, we outline the essential concepts and rationale behind the programming model that we use in Matisse. Detailed language syntax is beyond the scope of this paper.

From our application experience on a number of industrial broadband telecom network applications, we can conclude that behavior of these applications are best characterized

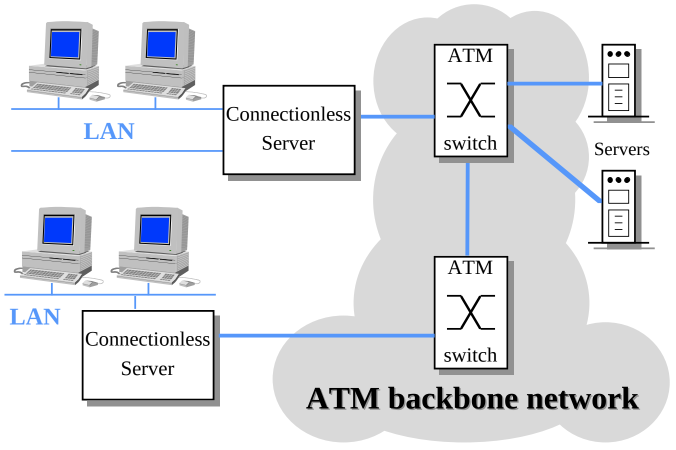


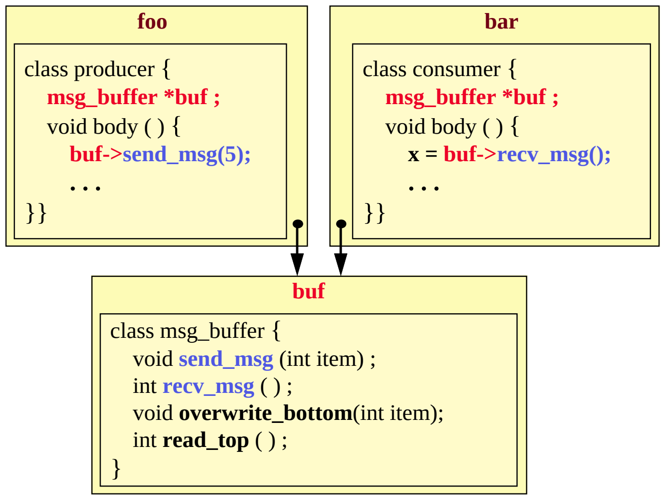
Figure 1: A user transparent connectionless router interconnecting LANs to the ATM network.

by *control-flow dominated data processing* algorithms. Expressiveness in terms of control flow is essential. The algorithms and the data structures that they operate on are usually tightly coupled. In many cases, the *data* represents the real conceptual core of the application rather than the algorithms. Support for concurrency is essential, but parallelism tends to be at the *task-level* and is usually coarse to medium scale. Although the implementation target is often a mixture of software and hardware, telecom network applications are often conceived at the top-level from a software perspective.

Given the above considerations, we have followed an object-oriented approach to parallelism as the basis of our programming model, as we believe it provides a natural model of concurrency for the applications considered: objects encapsulate processes and (remote) member function calls encapsulate interprocess communication. In addition, we believe object-oriented features like encapsulation, inheritance, and polymorphism are invaluable in any large scale development. Specifically, we use an *active object* semantic model that combines parallelism with object oriented principles. We have implemented the programming model on top of the widely used object-oriented programming language C++ [24] by introducing minimal syntactic extensions to it.

An active object differs from a passive object (as found in standard C++) in that it additionally encapsulates a process, as well as state and operations. The process allows an active object to execute its member functions in parallel with the activity of the rest of the program. In a typical program, only a small number of objects will be active, providing the parallel structure of the program. The majority of objects will be standard passive objects, existing as members of active objects or in data structures managed by active objects. This provides support for coarse to medium scale parallelism.

Communication between active objects is via the use of (remote) member function calls. A member function call to an active object behaves in the same way as a standard C++ mem-



```
class producer {
    msg_buffer *buf ;
    void body ( ) {
        buf->send_msg(5);
        . . .
}}
```

```
class consumer {
    msg_buffer *buf ;
    void body ( ) {
        x = buf->recv_msg();
        . . .
}}
```

```
class msg_buffer {
    void send_msg (int item) ;
    int recv_msg ( ) ;
    void overwrite_bottom(int item);
    int read_top ( ) ;
}
```

Figure 2: Simple example of model.

ber function call, in that the caller is suspended until the called member function terminates. This rendezvous semantics provides synchronization between active objects that are operating concurrently. Argument passing (in both directions) is the principle mechanism of data transfers. In addition to the basic notions of active objects, and communication and synchronization via (remote) member function calls, our programming model also provides mechanisms for mutual exclusion based on Hoare's monitor paradigm [15]. A simple example of the model is illustrated in Figure 2.

Since we have implemented our model on top of C++, the language also supports mechanisms for expressing (abstract) data types, assignment and arithmetic operations, and control-flow statements like if-then-else, switch-case, and for- and while- loops. Also, we can leverage upon the wide corpus of existing software compilation and runtime support tools for the software implementation path, and on existing execution environments [10, 12] and debugging tools [13] for early conceptual validation of the system specification.

In developing the model, the aim of maintaining as much compatibility with C++ as possible has been a central issue. We have avoided unnecessary differences from C++ where possible. This enables us to leverage on the wide corpus of existing software compilation and runtime support tools for our software implementation path. This is important since software implementation represents a substantial part of many of our target applications. In addition, compatibility with C++ enables new users already familiar with C++ to be productive in a very short amount of time. Also, existing execution environments and debugging tools can be easily adapted for early conceptual validation of the system specification.

While concurrent programming models based on object-oriented principles are not new, e.g. [4, 23, 1, 3, 7, 21], our aim here is different. These language environments typically assume elaborate runtime environments that can support dynamic process management, object migration between processors, network transparency, among other runtime features. Instead, our language environment must be suitable for embedded applications where the implementation may be partly in hardware and where software implementations are assumed to be supported by ultra-small real-time kernels that only offer bare minimum support for task scheduling and interprocess communication. These *microkernels* can be less than 3K byte in code size, in contrast to distributed runtime environments that are many orders of magnitude larger. The software implementation part of the design flow is outlined in Section 4.4, and the hardware implementation path is outlined in Section 4.5.

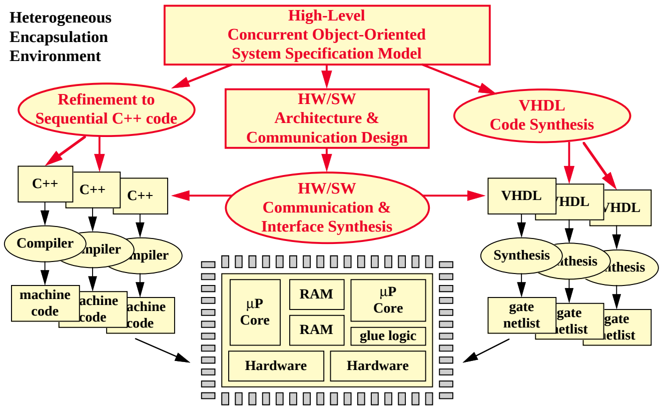

Heterogeneous Encapsulation Environment
High-Level Concurrent Object-Oriented System Specification Model
Refinement to Sequential C++ code
HW/SW Architecture & Communication Design
VHDL Code Synthesis
C++
C++
C++
HW/SW Communication & Interface Synthesis
VHDL
VHDL
VHDL
Compiler
Compiler
Compiler
Synthesis
Synthesis
Synthesis
machine code
machine code
machine code
µP Core
RAM
RAM
µP Core
glue logic
Hardware
Hardware
gate netlist
gate netlist
gate netlist

Figure 3: The Matisse system design flow in the CoWare environment.

4 The Design Methodology

The programming model described in the previous section provides the system designers with a specification formalism that can be used to specify the system level behavior of typical telecom network applications. This allows designers to use a formal specification formalism instead of informal report documents as a medium of exchange to lower level implementation paths. In this section, we describe the design tools that are a part of our design flow from system specification to implementation. The design flow is shown in Figure 3. Section 4.1 describes our simulation and debugging strategy. Section 4.2 describes a target abstract machine model for implementation. The concurrent specification model is implemented on this target machine model by partitioning it into software and hardware parts. Section 4.3 describes our view on software/hardware partitioning. Starting from the partitioning, conventional C++ modules are automatically generated for the software parts for entry to existing C++ compilers and VHDL code modules are automatically synthesized for the hardware parts for entry to existing VHDL-based hardware synthesis tools. Section 4.4 describes the software generation step, and Section 4.5 describes the VHDL code synthesis step. Section 4.6 outlines how the hardware/software communication is implemented. Finally, Section 4.7 outlines the role of a heterogeneous design environment for capturing the different stages of system design.

4.1 Simulation and Debugging

As our specification formalism is based on minimal syntactic extensions to C++, a wide corpus of existing software development tools may be leveraged. For simulation, we make use of existing runtime environments like PVM [10] and Nexus [12] that provide a transparent runtime layer above conventional operating systems like UNIX[1] for execution of distributed programs across a network of possibly heterogeneous workstations. Support for simulation and early validation at high levels of abstractions is essential in order to enable increased productivity and competitiveness. Augmenting the execution environment are debugging tools. Programs described in our model may be debugged by the same debuggers that are used to debug standard C++ programs. Existing debuggers like `dbx` and GNU's `gdb` are thread-aware. These debuggers can be used in conjunction with runtime layers like PVM and Nexus with the necessary preprocessing. In addition, analysis tools like profilers can be similarly adapted. We use these tools in our design flow for system validation.

4.2 Target Abstract Machine Model

This section describes the underlying target abstract machine model of Matisse. The implementation model is based on an abstract machine consisting of a networked collection of virtual processing objects, each being a processor/memory pair. This model is depicted in Figure 4. The model assumes that each virtual processor supports only a single active object. All data storage (e.g. for data structures) for an active object is mapped to the local (virtual) memory of the corresponding virtual processing object. This local memory model corresponds very closely to the notion of data encapsulation in the object model. Communication between virtual processing objects occurs via messages that are communicated over an abstract communication network. This communication network is implemented differently depending on whether the virtual processing objects are mapped to software or hardware, and whether several virtual processing objects are mapped to the same physical software processor. This abstract machine model provides an intermediate abstraction for reasoning about the relationship between the active object programming model and the eventual implementation.

The mapping of virtual processing objects to real processors depends on whether the implementation is in software or in hardware. In the case of software, several virtual processing objects can be mapped on to the same physical software processor and the same physical local memory. A ultralightweight real-time operating system kernel is used to manage these tasks and the memory partitions. In the case of hardware, each virtual processing object is in fact mapped physically to a separate hardware processor with its own physical local memory. This is because hardware is inherently parallel. For the communication network, we currently implemented dedicated point-to-point channels between physical processors. The implementation of the low level channel hardware is described in [19].

[1] Trademark of AT&T Bell Laboratories.

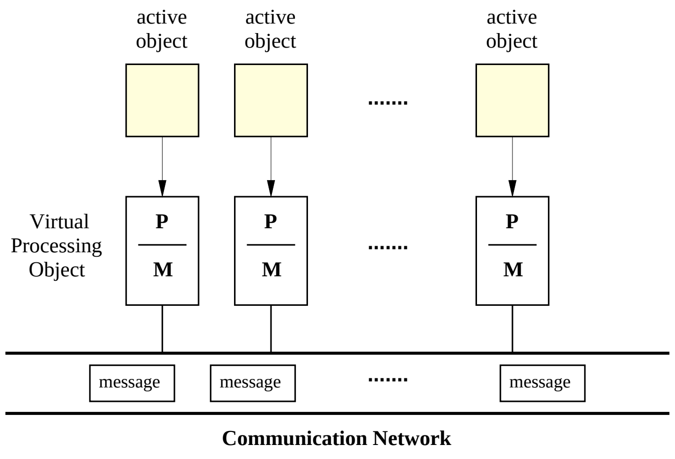


Figure 4: A target abstract machine model.

4.3 Software/hardware partitioning

Software/hardware partitioning is driven by the designer by means of directives. System partitioning decisions are driven by factors that are often not easily quantifiable. These factors often cannot be easily formulated into simple cost equations that an automated tool can optimize. Therefore, we believe that such decisions should be best left to the designer who is in a better position to make such judgements. We instead try to automate the refinement steps to lower level hardware and software tools as these steps tend to be the most problematic for designers.

4.4 Software implementation

For software implementation, we use existing C++ compilers to produce machine code for the embedded processor target. However, these compilers are aimed at compiling conventional C++ code, which is sequential. To manage concurrent tasks running on the same processor, we make use of ultralightweight real-time microkernels, which provide bare minimum services for task scheduling and interprocess communication. These services are provided at the level of library function calls. Currently, we make use of a solution based on a commercial real-time kernel called Virtuoso [27, 20].

To make use of such real-time microkernels and conventional C++ compilers, automated compilation tools are provided in our design flow to transform our concurrent specification model to a level of design entry suitable for these tools.

4.5 Hardware implementation

Starting from system specification model, refinement to lower level hardware and software design frameworks is automated. Designers will be able to start from the same system specification model instead of having to re-specify and re-interpret the system specification into lower level hardware description languages like VHDL. This will help to eliminate time and effort spent in understanding and interpreting an informal specification, which is again often susceptible to ambiguity and mismatches.

Specifically, we use behavioral VHDL as the level of design entry to current commercial hardware synthesis tools. This VHDL is automatically generated from the parts of the system specification that are assigned to custom hardware. Several first generation commercial behavioral synthesis tools are appearing on the market. Examples include the Behavioral Compiler from Synopsys and Mistral-2 DSP Compiler from Mentor. These compilers incorporate many of the key results from the last decade of high-level synthesis research (see e.g. [6]). We are currently experimenting with the Synopsys Behavioral Compiler [25] as the backend in our design flow, which already provides basic functionalities like dependency graph construction, scheduling, resource and register assignment, and datapath and controller synthesis.

Despite the availability of these high-level synthesis functionalities, there is still a significant gap from our system level model to the behavioral VHDL model. The object model has to be translated into VHDL's package and entity models. Also, provisions for dynamic data structures inherent in the object-oriented programming model must be supported. Behavioral synthesis tools today only support simple data types like static records and arrays. Dynamic data structures are not supported. Instead, the designer has to explicitly map them in terms of memory locations and memory operations. Hardware behavior must be explicitly provided to manage the runtime behavior of the memory. For applications in the telecom network domain, this can be a significant limitation. Moreoever, the abstract notion of data structures and having algorithms operate on them, the basic premise of object-oriented programming, is partially lost. In our refinement step to behavioral VHDL, we automate the allocation of physical memory, the mapping of dynamic data structures to memory locations, the refinement of data structure accesses to primitive memory operations, and the synthesis of dynamic memory management behavior for the runtime management of the memory. Some of these issues have been addressed in [11].

With these additional hardware synthesis functionalities, the designer can start from the same level of design entry as the system specification. This means changes at the system level specification are easily accommodated since the compilation to lower level hardware design steps is automated.

4.6 Software/hardware communication synthesis

Different parts of the specification will be assigned to software or hardware. Since we are building an application-specific solution, different software programmable processors from different vendors may be used, thus resulting in a heterogeneous hardware/software architecture. The implementation of such heterogeneous embedded architectures, while ensuring that the different system components are correctly integrated together, is a surprisingly difficult task. Designers spend an enormous amount of time on this task, partly in understanding how to interface to the different processors being used, how to get the hardware and software parts to communicate correctly, and how to synchronize between different components operating on different clocks. This is a highly error proned task, often responsible for many low-level implementation mismatches, leading to a length test phase after implementation.

Our approach to this problem is based on an orchestrated combination of architectural strategies, parameterized libraries, and CAD tools for automating low-level design tasks that are error proned and time consuming. More specifically, tools are provided to implement the *communication structure* between the software and hardware components, thus solving the system integration and architecture co-implementation problems. The communication structure will support point-to-point communication, shared bus based communication, and shared memory based communication. In addition, the communication and architecture synthesis tools will also support the *embedding* of real-time kernels for multi-tasking support on a software processor. Although there are many existing commercial real-time kernels available, they are not readily portable to application-specific embedded architectures. To solve this problem, tools are also provided to automate the low-level configuration of real-time kernels so that they can be used with the communication structures and embedded architecture that we generate. These tools are embodied in a system called *Symphony* [19, 20].

4.7 Heterogeneous co-design and co-simulation

Based on a common system specification model for capturing control-flow dominated data processing applications, lower level hardware and software specification models are generated for further synthesis. This is complementary to hardware/software co-design flows from system data flow models for DSP applications (e.g. [17]), which are also based on a common top-level representation. However, there are design situations where it would be best to mix different formal system design models. This is one manifestation of heterogeneity.

Even for co-design flows where a common formal system design model is used, such as the one described here, heterogeneity will naturally manifest itself by the virtue of the fact that different *implementation technologies* will be used. For example, C/C++ models will most likely be used for software targets, and VHDL/Verilog models will most likely be used as a lower level design entry for hardware targets. In general, there are many immediate steps from system specification to final implementation, and different computational models are required at different levels of the refinement trajectory.

Thus, a *heterogeneous* co-design environment is needed that can integrate multiple computational domains at different levels of abstraction within a single environment. We are currently developing one such environment called CoWare [8]. Ptolemy [5] is another such environment. A key problem is the encapsulation of different design tools. For example, the Matisse system described in this paper, the Symphony system for hardware/software communication, the Synopsys compiler, and C++ compilers for different processors used, are all being integrated into our CoWare environment.

5 Related Work

Data flow models have been successfully used to model DSP-oriented systems (e.g. the SDF/DDF domains in Ptolemy [5] and the Grape2 system [18]). Commercial offerings include

SPW from Alta/Cadence, COSSAP from Synopsys, and DSP-Station from Mentor. While sophisticated compilers have been developed for mapping these models into hardware and software, they are not well suited for control-dominated data processing behaviors found in telecom network management applications that heavily rely on tight interactions between control-flow algorithms and stored data structures (e.g. the case study in Section 2).

Hierarchical FSM models, as exemplified in commercial systems like Statemate from iLogix, have proved to be a powerful formalism for reactive control behaviors. These models do not support well programming constructs like abstract data structures and do not support object-oriented features.

Distributed programming languages have been proposed for programming general-purpose multiprocessor systems or a distributed network of workstations [4, 23, 1, 3, 7, 21, 10, 12]. While the programming models are strongly related to the one we use in this work, their implementation targets are different. They rely on elaborate runtime environments for their execution, and they are intended for pure software implementations. In contrast, our implementation target is an embedded solution that is designed and optimized to provide specific functionality for a particular application, using a combinational of both software and hardware components. Often single chip solutions are sought. This target has a significant impact on the design flow and methodology.

Heterogeneous design environments, like the Ptolemy system [5] and the CoWare system [8] under development at IMEC, aim to provide an open environment where different models of computation, and their accompanying simulation and design methods, can be smoothly integrated. The work presented here is complementary as it can be embedded into such heterogeneous environments so that it can be used in combination with other design models and methods. In fact, the work presented in this paper is being incorporated into the CoWare design environment at IMEC.

6 Conclusion

We have presented in this paper a system design methodology for the co-development of hybrid software/hardware systems for telecom network applications. This methodology was based on the results of an investigation and evaluation of an actual industrial system design application for ATM-based broadband networks. Specifically, the case study of a user transparent connectionless router for ATM has been investigated. The proposed methodology is based on a concurrent object-oriented programming model, and we have presented new design steps that are a part of an integrated design flow from system specification to implementation. In the future, we plan to further validate our methodology and design tools on other relevant industrial design applications in the area of telecom networks.

Acknowledgments. This work is part of a joint collaboration between IMEC and Alcatel-Bell. The author would like to thank M. Genoe, G. Van Wauwe, E. Huyskens, and L. Cloetens from Alcatel-Bell for their contributions and support in this project. The work outlined in this paper are the result of ongoing research work at IMEC, to which many people are contributing. In particular, the author is grateful to the contributions from G. de Jong, J. Silva, S. Vercauteren, and C. Ykman.

References

[1] H. E. Bal, M. F. Kaashock, and A. S. Tanenbaum. “Orca: A Language for Parallel Programming of Distributed Systems”, *IEEE Transactions on Software Engineering*, 18(3), March 1992.

[2] Bellcore, “Generic System Requirements in Support of Switched Multi-megabit Data Service”. TR-TSV-0007772, n^o 1, May 1991.

[3] A. Black, N. Hutchison, E. Jul, and H. Levy. “Object Structure in the Emerald System”, In *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 78-86, 1986.

[4] G. Booch, *Object-Oriented Design with Applications*, Menlo Park, CA, Benjamin/Cummings, 1991.

[5] J. T. Buck et al. “Ptolemy: A framework for simulating and prototyping heterogeneous systems”, *International Journal on Computer Simulation*, January 1994.

[6] R. Camposano and W. Wolf (editors), *Trends in High-Level Synthesis*, Kluwer Academic Publishers, 1993.

[7] D. Caromel. “Toward a Method of Object-Oriented Concurrent Programming”, *Communications of the ACM*, September 1993.

[8] H. De Man, I. Bolsens, B. Lin, K. Van Rompaey, S. Vercauteren, and D. Verkest. “Co-design of DSP systems”, *NATO ASI Hardware/Software Co-Design*, Tremezzo, June 1995.

[9] M. De Prycker, “Asynchronous Transfer Mode, Solution for Broadband ISDN”, Ellis Horwood, 1991.

[10] J. Dogarra, G. Geist, R. Manchek, and V. Sunderam. “Integrated PVM Framework Supports Heterogeneous Network Computing”, In *Computers in Physics*, April 1993.

[11] G. de Jong, B. Lin, C. Verdonck, S. Wuytack, F. Catthoor, “Background Memory Management for Dynamic Data Structure Intensive Processing Systems”, *ICCAD*, November 1995.

[12] I. Foster, C. Kesselman, S. Tuecke, “Nexus: Runtime Support for Task-Parallel Programming Languages”, Technical report, Mathematics and Computer Science Division, Argonne National Laboratory, 1994.

[13] GNU distribution. Free Software Foundation. 1994.

[14] D. Harel. “Statecharts: A Visual Formalism for Complex Systems”, *Sci. Comput. Program*, vol.8, pp. 231-274, 1987.

[15] C.A.R. Hoare. “Monitors: An operating system structuring concept”, *Communications of the ACM*, 17(10):549-557, October 1974.

[16] “IEEE Standard VHDL Language Reference Manual”, IEEE Std. 1076-1987, IEEE, New York, NY, 1988.

[17] A. Kalavade, E. A. Lee. “A Hardware/Software Codesign Methodology for DSP Applications”, *IEEE Design & Test*, Sept. 1993.

[18] R. Lauwereins et al. “Grape-II: A system level prototyping environment for DSP applications”, *IEEE Computer*, pp.35-43, February 1995.

[19] S. Vercauteren, B. Lin, H. De Man. “Constructing Application-Specific Heterogeneous Embedded Architectures from Custom HW/SW Applications”, *ACM/IEEE Design Automation Conference*, June, 1996.

[20] S. Vercauteren, B. Lin, H. De Man. “A Strategy for Real-Time Kernel Support in Application-Specific HW/SW Embedded Architectures”, *ACM/IEEE Design Automation Conference*, June, 1996.

[21] B. Meyer. “Systematic Concurrent Object-Oriented Programming”, *Communications of the ACM*, September 1993.

[22] “Mistral-2 Architecture Compiler”, Mentor Graphics, Beaverton, OR.

[23] J. Rumbaugh et al., *Object-oriented modeling and design*, Prentice-Hall, 1991.

[24] B. Stroustrup, “The C++ Programming Language”, Addison-Wesley, 1986.

[25] “Synopsys Behavioral Compiler”, Synopsys, Mountain View, CA.

[26] Y. Therasse, G. Petit, M. Delvaux, “VLSI architecture of a SMDS/ATM router”, in Annales des Télécommunications, 48, n^o 3-4, 1993.

[27] E. Verhulst, “Virtuoso, DSP programming tools covering from distributed multitasking to synchronous dataflow”, Eonics Inc., 1994.