



Publication

LIFL 2002-n°06

**Sophocles: Cyber-Enterprise for System-On-Chip
Distributed Simulation – Model Unification**

Pierre Boulet, Jean-Luc Dekeyser, Cédric Dumoulin, Philippe Kajfasz, Philippe
Marquet et Dominique Ragot

28 juin 2002

Sophocles*: Cyber-Enterprise for System-On-Chip Distributed Simulation – Model Unification

Pierre Boulet[†] Jean-Luc Dekeyser[†] Cédric Dumoulin[†] Philippe Kajfasz[‡]
Philippe Marquet[†] Dominique Ragot[‡]

[†]Laboratoire d'Informatique Fondamentale de Lille
Université des Sciences et Technologies de Lille
France

[‡]Thales Communications
Gennevilliers
France

Abstract

Complexity in the digital systems integration rises from the heterogeneity of the components integrated in a chip. The aim of the Sophocles project is to validate methodologies, platforms and technologies to support integration, verification and programming, over a distributed environment, of complex systems composed of heterogeneous virtual components. Several formalisms are gathered, according to their applicability, in order to immediately propose a framework of formal specification and validation of applications for systems-on-chip. The unification of these formalisms in a modeling language facilitates the work of the users while guaranteeing a strong semantics on all the levels of the specification.

1. Introduction

In the next decade, “Software driven systems” will play a crucial role in telecommunication (3rd/4th-generation mobile phone, both terminals and base stations...) and multimedia applications (set top box, multimedia computers...). These systems will have to deal with all the various software programming levels like intensive processing (i.e. digital signal processing, data compression...), image processing and/or communications (i.e. protocol stacks...), and require system-level programming environments.

The architecture of these systems will be fundamentally heterogeneous. It will be based on the integration on systems-on-chip of various computing engines devoted to specific functions like intensive processing, data processing and/or decision and supervision.

Unfortunately, programming and performance estimation of such digital systems becomes more and more complex. Real-time application development on this kind of engines, closely interconnected (sometimes on a single silicon die), is not trivial at all. Complexity of software development (processing and communication tasks) is drastically increased mainly because these on-chip inter-processors buses are sometimes not accessible and often not properly modeled.

So, in the next years, to be able to implement such applications, it will be necessary to use high level programming environments coupled to efficient heterogeneous cycle-accurate simulators and global system modeling.

Presently, for each new system design, new tools and new simulators have to be developed, starting from proprietary models with no communication standards and/or capabilities.

The aim of the Sophocles project is to reach a conceptual validation of methodologies, platforms and technologies supporting the integration, validation and programming, over a distributed environment, of complex systems composed of heterogeneous virtual components.

This methodology should permit the birth of cyber-enterprises devoted to provide, over the Web, the integration services. Main users of the methodology will be complex system architects, virtual component providers, Intellectual property designers and final system producers. This cyber-enterprise methodology will greatly benefit during all the stages of integration, validation and programming, from advanced cognitive interfaces.

Cognitive interfaces could provide clever support to the activities of system architects. The environment could be integrated with multimedia user interface, permitting a multi sensorial contact with the charac-

*Sophocles is the european ITEA project number 99038.

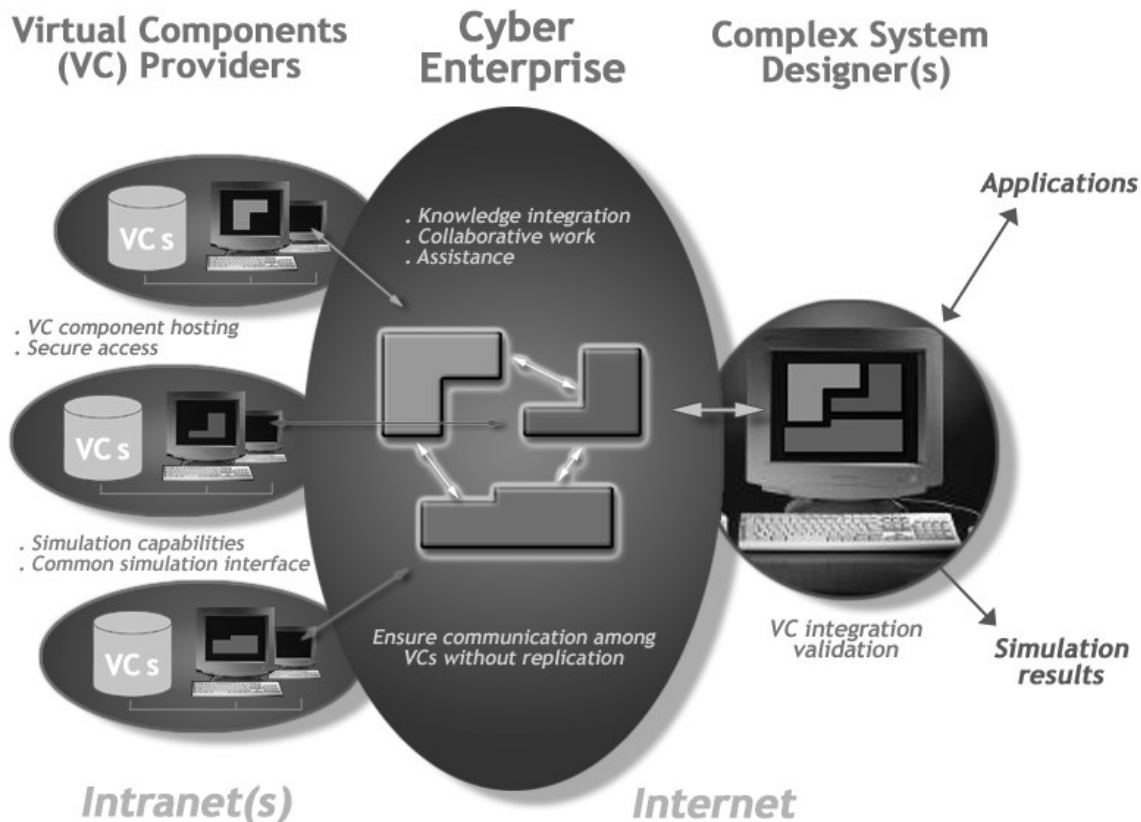


Figure 1. Cyber-enterprise

teristic of the system under assembly.

We will first present the cyber-enterprise conception in the next Section, then in Section 3 we discuss the need of high-level formalisms to model applications, architectures and the mapping of applications on architectures. In Section 4 we state how UML can be a common language for all these specifications.

2. The Cyber-Enterprise

The advances in digital system integration are visible in two directions: increasing complexity at macro-scale (integration of heterogeneous digital components for data acquisition/supercomputing applications) design using heterogeneous blocks at chip-level scale.

The implementation of an application on such systems requires actions to be taken at high-level to determine as early as possible in the design process what alternatives are viable.

The use of components, though widespread in both software and hardware worlds, is facing a major chal-

lenge of complexity management. Usually, components are available as basic blocs for design of systems, or as a whole set of functionality. In either case, the black box model prevails, and there is no point in getting access to the component structure. Combining components in such a case is apparently easy, but in practice difficulty arises from the fact there is no precise description of the semantics of interaction between components.

The top-down approach, by designing models, is often limited by the fact that the model itself implements a single semantic of interactions, thus making it difficult to represent a system with its global behavior. The solution in this case is to develop a number of models, each accounting for a given formalism. It is hence very difficult to assess that these multiple models of a given system are globally coherent, and that they indeed represent the effective behavior of the system.

In the world of virtual components, there are an increasingly large number of blocks accessible to the application developer. The primary focus of these blocks is towards implementation, but some have high-level

models available. For applications that are using a large number of these blocs, the complexity resides in the management of the interactions between them. Just assembling models of components simply does not work, in the sense that it can not provide useful system-level information to the application developer.

The primary goal of the Sophocles project is to use a set of high-level formalisms to describe the semantics of interactions between virtual components and to bring to the application developer a set of methods and tools to allow a high level description of a system.

If simulation is aimed at solving issues about the management of complexity in the early stages of the designs, there are two main problems to be faced: on a technical standpoint, simulation of complex systems will require more computational power as the system complexity or the level of detail increases; on a business standpoint, the cost of possession of simulation software and of the simulation infrastructure is far beyond the investment capabilities of many mid-sized companies. Moreover, the shorter application lifetime, and time-to-market issues, are mostly incompatible with the classical invest-deploy-use for these type of tools.

The solution studied in Sophocles is based on the cyber-enterprise (Figure 1) where companies would not necessarily have to possess their simulation infrastructure and tools, but could rely on third-party companies whose specific business is to provide for a number of client companies a set of virtual components (bought from virtual component providers) and a computing infrastructure capable of hosting part of simulations using these components.

3. High-level Formalisms

In the Sophocles project, the specification of the algorithms at the base of any simulation on systems-on-chip is built on several formalisms. Each one answers a precise part of the specification: algorithm, architecture, mapping and scheduling (Figure 2). The use of these formalisms is of three distinct and essential interests in real time application description and in their simulation on systems-on-chip.

An early complete functional specification. From visual modeling tools, we propose a non-ambiguous notation associated with a formal semantic that applies on several levels of the applications: visual model for the specification of the algorithm, the architecture of an embedded system and a deployment of the algorithm on this architecture.

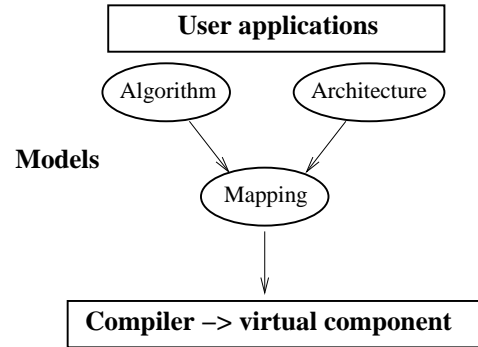


Figure 2. High-level specification models - The “Y” architecture

Validation, performance evaluation and verification. We support the formal verification of the functionalities of applications by automatic generation of tests, the evaluation of the performances starting from the description of the algorithm and its mapping on a modeled architecture. The goal remains the simulation on an assembly of virtual components by the means of the cyber-enterprise.

Separate different levels of application description. The model separation offers re-use facilities. The same architecture or application should appear in several projects, in particular to re-use an application on a new architecture, to develop a new application or to transform an application on an existing architecture. Of course the mapping remains related to the application and the architecture. We propose a visual specification environment for this “Y” architecture (Figure 2) using a unique metaphor. It must facilitate the exchange between the various models without requiring the mastering of several languages. Each model owns its methodology, so the interactions between the three models imply the definition of collaborative and coherent methodologies. To ensure coherence between the models, we can extract a certain number of rules:

- To use the same formalism: a user must be able to go easily from a specification to another without having to learn new concepts. This concerns the expression of the dependencies: space and temporal data for the applications, of data flow during the time for the material components.
- To use the same notation: here we retain the “visual language” UML. We add extensions and construction rules. These extensions and these rules,

initially, are joined together in a UML profile dedicated to the Intensive Signal Processing (ISP). Thereafter we plan to provide a language derived from UML and dedicated to the ISP. This language will be expressed using the MOF.

- To use the same internal representation: the visual tools and the exploitation tools of the models use internal representations (memory or persistent representations). We propose a common representation to all these tools we develop.
- To use the same external tools: the used tools must suite to the various specifications.
- To ensure an automatic exploitation: various methodologies will allow to automatically obtain application models usable by various tools such as code generators, simulators...

3.1. Application specification

From the observation of various specification models, we deduced that intensive signal processing specification model has to respect the following constraints.

- Single-assignment: the data are mainly arrays. They are produced by an task once and consumed by other tasks. This single-assignment of arrays facilitates the visual specification of an application.
- Unification of temporal and space dimensions: dimensions of the arrays are mainly characteristics of the application (hydrophones, sensors, energy...). One dimension can be associated to the time, it then allows the identification of this various values during the life of the application. This dimension becomes of infinite size for an embedded application.
- Expression of the temporal and space dependencies: it represents the only link between the various objects handled by the program. It provides the dependencies between the elements of the objects (arrays). It covers as much space dimensions as temporal ones.
- Universality of a programming language: the applicability can easily be extended to other fields that signal processing. An intensive data processing is often associated with the systematic signal processing. This intensive data processing is often irregular, it handles dynamic data structures and the processing is done with variable behavior. In

order to propose a single model for these two successive types of specification, we add two essential characteristics: the recursivity, it ensures to the traditional systems of signal processing the power of the recursive languages; the first class citizen components, that increases the application area of our model by bringing the universality of the functional languages.

3.2. Data Dependence Expression

To have a high-level description of an algorithm we have chosen to express only the data dependences. Indeed, they completely describe the algorithm without addition of any parasite information. Actually any compilation optimization or parallelization technique must respect the data dependences. This gives us many benefits:

- simple description of the algorithm,
- no dependence analysis in the compiler,
- all the parallelism and optimization potential of the algorithm is easily available.

When dealing with real-time applications, one usually uses clocks to express time dependencies. To avoid these clocks and their added complexity, time is represented as a dimension in our data objects (mainly arrays). Dependences along this “temporal dimension” are equivalent to constraints on clocks.

The data dependences are specified on several levels: on a *global* level and on a *local* level. The global level expresses the dependences between the complex objects (arrays) manipulated by the components of the application while the local level expresses the finer dependences between the elements forming these objects.

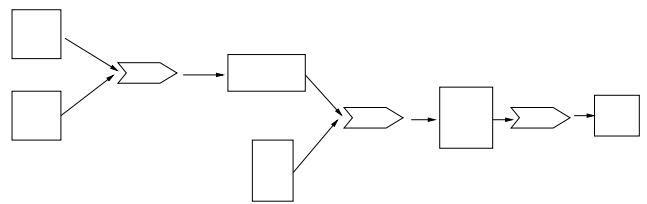


Figure 3. Example of a global model: a graph of tasks and data objects

The global level (Figure 3) is easily translated into a traditional task graph which exhibits task parallelism. Such a graph can be seen as a Kahn process network and can be translated into a YAPI [5] application or

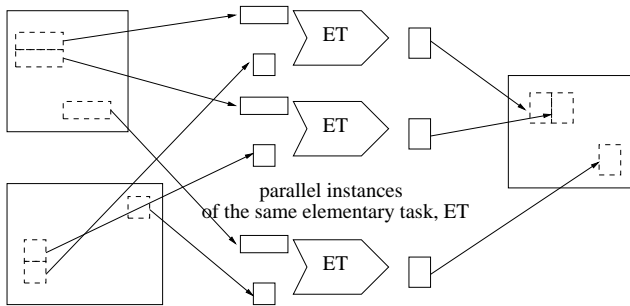


Figure 4. Example of a local model: each elementary task works on parts of the data objects

a distributed CORBA component graph [2]. From the local level (Figure 4) one can extract data parallelism that can be simulated with SMP computers and compiled to SIMD or DSP virtual components.

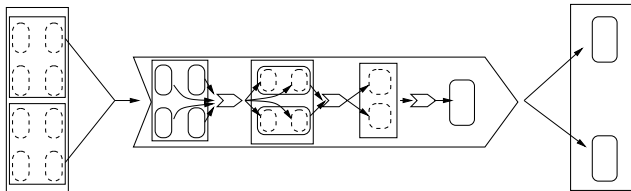


Figure 5. Example of hierarchical components

Further more, using the concept of *hierarchy* one can further modularize the algorithm specification. Figure 5 depicts such a hierarchy. Indeed, a component can be itself decomposed into subcomponents by a global or a local level. This allows to reuse component specifications and to progressively refine an application. The specification of an application can then be compact and parameterized.

The formalism we use inherits from the Array-OL language [3] which is specifically designed to specify intensive signal processing applications. All the data in Array-OL are arrays and the two-level dependence expression and the hierarchy are central in the language. One other interesting feature of Array-OL is that it describes the algorithms in a single-assignment form [4]. This assures that the only dependences are true data dependences and allows maximal optimization freedom.

We are currently extending Array-OL to handle more irregular applications by proposing different de-

pendence expressions such as an “enumerative” expression, allowing recursivity and adding different data types. This extension respects the main aspects of Array-OL: a two level dependence expression, the hierarchy and the single-assignment form. We believe that these extensions will give us the power to easily express all the computation intensive algorithms used in signal processing while keeping the benefits of a high level specification.

Finally all this specification is completely independent of the target architecture and can be used as is both for simulation and for compilation on a system-on-chip.

3.3. Architecture and Mapping Specification

Then we can specify a hardware architecture which corresponds to the particular target for the development of this application. There is a need for a high level formalism to specify systems-on-chip and the mapping of applications on systems-on-chip.

The architecture specification formalism should be modular and hierarchical to allow the description of systems-on-chip based on virtual component ones. A hierarchical model allows to identify processing elements, data transfer elements and the data paths between these elements. This description should be multi-levelled, from a purely functional specification to a bit accurate / cycle accurate description.

Two technologies are under development in the Sophocles project: MADE and Array-OL architecture. MADE [11] stands for Modular VLIW Processor Architecture Description Environment and allows to generate a custom assembler-optimizer from a high level description of a VLIW processor.

Array-OL architecture (ongoing project at Thales Underwater Systems) is based on the same concepts as the Array-OL algorithm specification language (Section 3.2):

- a two level specification,
- the hierarchy,
- arrays to represent data.

The global level represents the data paths between different units and the local level allows to represent regular units such as memory or SIMD computation cores. The elementary components can be either active components (computation units or DMAs) or passive units such as memory.

The mapping of an application onto an architecture is described as an association of the components from the algorithm specification to some components of the

architecture specification. This mapping is both space-wise and time-wise, some dimensions of the data arrays being mapped onto dimensions of the memory elements or schedule in temporal dimensions. Array-OL architecture proposes elegant means to achieve the placement of an Array-OL application. The mapping techniques are manual, semi-automatic, even automatic. From a full modeling of the low-level layers, the environment is able to generate the suitable code for a simulation or for an execution.

4. UML as a Common Specification Language

To model and express algorithms, architectures and mappings between both, we propose to use the Unified Modeling Language (UML) [8].

We choose to use UML for the following reasons:

- UML is recognized as a standard and is widely adopted by the industry.
- UML offers extension mechanisms ("stereotype", "tagged value", "profile") allowing us to implement our own language elements without modifying UML.
- UML does not depend on any particular methodology. We can define and validate our own methodology.
- UML is a visual as well as a textual language. It offers visual, multiple and hierarchical specification capabilities by the way of diagrams.
- Different efficient visual tools exist (Rational Rose, Objectteering...). The conformity of models exchanged between tools is ensured by the XMI / XML standard [10].

UML has been originally developed to model the artifacts of intensive software systems. This includes application modeling, architecture modeling and mapping. Therefore, it is possible to specify an intensive signal processing application with UML. However, UML doesn't fulfill all our requirements. There is no methodology allowing automatic exploitation of models. Sophocles needs a strong framework and guidelines to express models that can be exploited by automatic tools (code generation, transformations, visualization...). Existing UML diagrams, while sufficient, are not always the best fitted to some visual design required by Sophocles and ISP. Finally, the architecture description and deployment provided by UML are rudimentary. Sophocles' models need a finer grain.

Other projects in the area of real-time systems or embedded applications have also chosen UML as a modeling language (SDL[1] UML-RT [6], RT-UML [9], Embedded UML [7]). Our approach differs from previous projects as we want to model embedded applications with the help of dependency expressions rather than by exchanging signals or message passing.

UML use as a tool for complete code generation is not well known, but several current projects chose this approach. Automatic exploitation of models to generate complete code is possible only if the model description comes with strict modeling rules. In our case, automatic exploitation of a model is possible as we restrict ourselves to specification of intensive signal processing applications and we follow strict modeling rules that we define. These rules are part of the methodology that we propose for modeling intensive signal processing embedded applications.

This methodology is based on the notion of signal processing components (sp-component). The entire framework can use only UML constructs: stereotyped classes for sp-components, associations to express dependences, and class and collaboration diagrams to visually model applications (Figure 6).

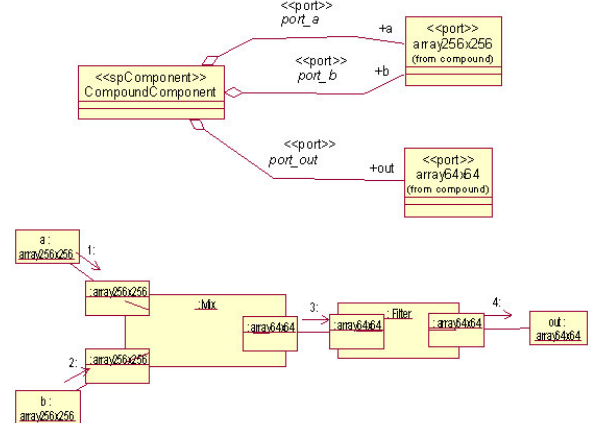


Figure 6. Compound sp-component modeling

Architecture and mapping modeling are done using a similar methodology.

The use of the UML unifies, in a common language, the specification of the applications, the architectures and the deployments and mappings. Furthermore, all these specifications can be done in one single integrated tool which is already available.

5. Conclusion

The environment we propose in Sophocles, through the cyber-enterprise, is to bring the needed infrastructure to build applications using virtual components and to execute these complex heterogeneous systems over the network. Currently, access to virtual components is done by specialized company catalogs or on the Internet, with limited virtual component descriptions and models. Most of the work done by companies and standardization bodies is bound by this bipolar view and virtual component distribution and integration in a system is not considered.

Sophocles provides application developers with an environment for the integration of applications at system-level using a combination of high-level formalisms to express data-flow and control via data dependences. UML is suggested to express these formalisms.

System-level designs are based on integration of virtual components (component plug and play) and may be executed in a distributed simulation fashion (possibly hosted by their providers). This allows a rapid design space exploration and an easy system validation and evaluation.

References

- [1] A. Alkhodre, J.-P. Babau, and J.-J. Schwarz. Preparing SDL code generation for real-time embedded systems modeling. In *IEEE/IEE Real-Time Embedded Systems Workshop (Satellite of the IEEE Real-Time Systems Symposium)*, London, Dec. 2001.
- [2] A. Amar, P. Boulet, and J.-L. Dekeyser. Assembling dynamic components for metacomputing using CORBA. In *Parallel Computing 2001*, Naples, Italy, Sept. 2001. Lecture Notes in Computer Science.
- [3] P. Boulet, J.-L. Dekeyser, J.-L. Levaire, P. Marquet, J. Soula, and A. Demeure. Visual data-parallel programming for signal processing applications. In *9th Euromicro Workshop on Parallel and Distributed Processing, PDP 2001*, pages 105–112, Mantova, Italy, Feb. 2001.
- [4] R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems*, 13(4):451–490, October 1991.
- [5] E. A. de Kock, G. Essink, W. J. M. Smits, P. van der Wolf, J.-Y. Brunel, W. M. Kruijtzter, P. Lieverse, and K. A. Vissers. YAPI: Application modeling for signal processing systems. In *37th Design Automation Conference*, Los Angeles, CA, June 2000. ACM Press.
- [6] R. Grosu, M. Broy, B. Selic, and G. Stefanescu. Towards a calculus for UML-RT specifications. In H. Kilov, B. Rumpe, and I. Simmonds, editors, *Seventh OOPSLA Workshop on Behavioral Semantics of OO Business and System Specifications*, Vancouver, Canada, Oct. 1998.
- [7] G. Martin, L. Lavagno, and J.-L. Guerin. Embedded UML: a merger of real-time uml and co-design. <http://www.gigascale.org/pubs/101.html>, Mar. 2001.
- [8] Object Management Group, Inc., editor. *Unified Modeling Language (UML), Version 1.4*. <http://www.omg.org/technology/documents/formal/uml.htm>, Sept. 2001.
- [9] Object Management Group, Inc., editor. *(UML) Profile for Schedulability, Performance, and Time Specification*. <http://www.omg.org/cgi-bin/doc?ptc/2002-03-02/>, May 2002.
- [10] Object Management Group, Inc., editor. *XML Metadata Interchange (XMI), Version 1.2*. <http://www.omg.org/technology/documents/formal/xmi.htm>, Jan. 2002.
- [11] P. S. Paolucci, P. Kajfasz, P. Bonnot, B. Candaele, D. Maufruid, E. Pastorelli, A. Ricciardi, Y. Fusella, and E. Guarino. mAgic-FPU and MADE: A customizable VLIW core and the modular vliw processor architecture description environment. In *SIMAI 2000 Conference*, Ischia, Italy, June 2000.