

# HIERARCHICAL MIXED SIMULATION FOR INTELLIGENT INTERFACES OF MICROSYSTEMS

Tudor NICULIU

Sorin COTOFANA \*

Anton MANOLESCU

Universitatea "Politehnica" Bucuresti, Facultatea de Electronica si Telecomunicatii

Bd. Iuliu Maniu 1-3, 77202 Bucuresti, Romania, E-mail: tudor@messnet.pub.ro

(\*)Delft University of Technology, Faculty of Electrical Engineering

Mekelweg 4, 2600 GA Delft, The Netherlands

## SUMMARY

The development of microsystems imposes hierarchical CAD procedures to manage their complexity. In this paper we attempt to handle microsystem intelligent interfaces by unifying representations for design and verification, as well as for different domains: electrical/non-electrical, analog/digital, hardware/software. Extending the functionality of an intelligent interface from I/O to internal collaboration between complementary domains, we propose an object-oriented framework to support domain specific methods, at different abstraction levels.

## 1. ARGUMENT

An object oriented approach addresses two main issues: representation and simulation-goal.

Representation is a mapping from the system universe to the model universe. Each model has two sides: one oriented to understand the system universe (description), and another patterned to the simulation operations (simulation). A generic modeling schema defines the universe of models, e.g., a mathematical theory or a programming paradigm, where each entity is characterized by: behavior (its relation to other entities, specified by its properties) and structure (its internal relations, specified by its parameters). The behavior can be expressed functionally or procedurally.

The simulation-goal guides the operations of the model. It can be either design (the structural parameters of the model are determined to obtain certain behavioral properties) or verification (the behavioral properties are deduced from structural parameters to be compared with the specification).

The development of Computer Aided Design (CAD) for microsystems calls for a combination of simulation techniques from different domains, e.g., as hardware/ software, analog/ digital, electrical/ non-electrical, as graphically depicted in Figure 1.

The complexity of the simulation's object imposes a hierarchical approach. Considering the heterogeneous relations between different functions that collaborate to build the behavior of the system, we have to extend the scope of intelligent interfaces, from standard I/O functions to more sophisticated conversions.

The intelligent interface concept implies integration of I/O functions to signal processing operations, parallel and distributed realization of

processing - to minimize communication problems, analog signal processing next to the interface, capability of choice among different precisions for an algorithm, easy and reliable calibration strategies, formal correspondence between different abstraction levels for different hierarchy types.

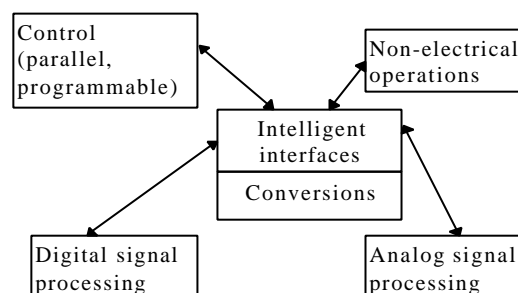


Figure 1: Functional view of a microsystem

We attempt to prove constructively that all these problems associated with the intelligent interfaces design and verification, can be solved in a knowledge-based object-oriented simulation framework. The paper presents the principles of the hierarchical approach and the architecture of a corresponding framework.

## 2. HIERARCHICAL SIMULATION

Hierarchies are leveled structures, which represent different domains. A level is an autonomous mathematical structure, containing abstract/concrete entities, linked by intralevel relations. Abstraction relates the levels: this induces an interlevel order relation, partial, concerning entities, and total, regarding the levels. Beyond the hierarchical point of view, the system can be formalized as an autonomous domain, structured by

meta-hierarchical relations, building a level in a higher order hierarchical system. Hierarchical structures exhibit two complementary processing strategies: top-down and bottom-up [Niculiu98].

Generally, multiple, interdependent hierarchies structure the universe of models. They belong to different hierarchy types, whose study and formalization result in definition of "basic" hierarchy types, that can be interpreted as knowledge-based [Winston92] and object-oriented paradigms [Booch91]:

- Simulation hierarchy: autonomous levels corresponding to different abstraction grades of representation;
- Structure hierarchy: recursive decomposition in autonomous blocks;
- Class-instance hierarchy: framework to represent all kinds of hierarchies, based on form-content complementarily, modularity, polymorphism, inheritance; an object contains data and operations; it is instance of a class and has an internal structure, defining its external behavior;
- Symbolization hierarchy: relation that can be viewed either as a particular form-content hierarchy or as a kind of knowledge hierarchy;
- Knowledge hierarchy: reflexive abstraction (different from the simplifying abstraction that corresponds to the other hierarchy types) represents the meta-hierarchical relations of the other hierarchy types, trying to extend the formalization of intelligence.

The different kinds of hierarchy correspond to mathematically structured object classes. Constructive type theory guides formal specification and proof of its correctness by generating objects that meet the specification.

### 3. OBJECT-ORIENTED REPRESENTATION

The design framework we propose has knowledge-based behavior and object-oriented structure. The design scenario, as it can be easily deduced from the overall architecture depicted in Figure 2, is as follows: Based on man-machine dialog, the simulation system can learn representations and solving strategies, then represent specifications (about facts, goals and relations). The resulted knowledge-base can guide model-generation and the codesign process. This is recursive, following the various hierarchies characterizing the simulated system, and iterative at each recursion step, to satisfy the constraints. The verification tools decide when to stop the iteration.

The primitives' library and method properties for various models and goals, as well as the abstraction levels for different hierarchies, the applicable

methods of analysis/ simulation and the evaluation strategies, constitute the object-oriented knowledge base [Weiler95] of the framework.

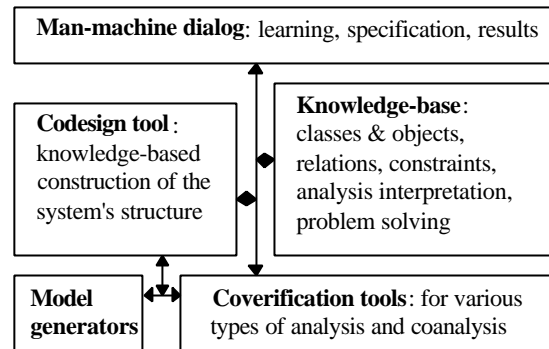


Figure 2: Object-oriented framework

The method properties contain: performance/cost of simple methods, characteristics of compound statements, lists of methods that can be executed in parallel, together with possible constraints (time, space, power, collaboration between parallel methods of the same object, external conditions). Codesign and coverification regard the various mixed simulation parts and will be further discussed in Section 5, 6 and 7.

### 4. INTELLIGENT SIMULATION

Knowledge-based architecture separates reasoning from representation. A system capable of reflexive abstraction («intelligent») reasons controlled by the problem specification and by solving strategies. These are derived from a higher level of knowledge, representing principles of approach, which are structured by an even higher level, containing hierarchy types. An object-oriented simulation framework permits the representation of different knowledge levels, each having a concept hierarchy, possibly abstraction/ structure/ symbolization leveled.

For representation, this approach offers the advantage of open modeling. Models are described by the user, following a general accepted paradigm, e.g., entity-architecture decomposition, which ensures syntactic correctness, leaving the meaning to be specified by user-defined semantic functions that control the simulation. For example, a module in an unfinished design can be characterized by constraints regarding its interaction to other modules; the constraints' model is open to be interpreted, thus implemented, in different ways that derive from adequacy criteria.

Explanation is a key concept for intelligent systems. It can be expressed as a proof in a deductive system, whose axioms are the equations

constraining component models and input signals, theorems are simulation results, inference rules represent logic & domain-specific calculus. Using constructivism in logic (e.g., intuitionist predicate logic), behavior or structure of the simulated system can be extracted from the proof [Bibel93].

Knowledge levels are mathematical types, strategies, simulation concepts, and application theories. Knowledge is represented oriented, as a hierarchical associative net of concepts. The upper levels contain functions defined on the lower ones. Interlevel relations can be viewed as: planning (top-down) and learning (bottom-up).

Planning transforms declarative knowledge (formal, but limited) in partially procedural knowledge (unlimited, but implying a context with resources that are not formalized at the upper level, the main being time). Artificial intelligence studies planning as reasoning about actions; actions are elements of a lower level, generally represented by states (instances of upper level functions in the presence of a context) and operations (determining state transitions). The plan is a non-commutative system of declarative knowledge.

Learning derives a formal structure on an upper level (e.g., static structure), from experiences on a lower level (procedures executed using resources that are not present at the upper level, e.g., time). It has two complementary aspects:

- 1) Induction: extensive knowledge at the lower level is transformed into intensive knowledge at the upper level, using non-reflexive abstraction (equivalence, isolation, emphasis, stationary approximation, idealization);
- 2) Deduction: intralevel relations produce new concepts (conditioning, association, imitation).

The modularity that characterizes hierarchical simulation can be extended to higher knowledge levels, expressing knowledge as generic behavior (templates), typifying objects (classes), managing simulation knowledge (packages, libraries).

## 5. HARD/ SOFT COSIMULATION

The hardware-software cosimulation of complex systems is imposed by the incompatibility or, at least, suboptimality associated with the initial hardware/ software partition of a design, and by the inefficiency of the design-verification cycle in the context of a fixed partition. To unify simulation methodologies, we started from the results of different research directions: object-oriented hardware/ software description, formal verification of software/ hardware, automated synthesis of hardware systems.

A unified representation for hardware and

software allows techniques from one domain to be applied to the other domain. Therefore, a representation based on abstraction and object-orientation, used primarily for software, is employed for the hardware domain as well. Also, existing software techniques, such as those used for verification of abstract data type implementations [Turner91], can be used for hardware.

```

void codesign (const Libraries* primitives,
              Hierarchies* abstraction_level,
              Properties* method_properties) {
    specification (abstraction_level);
    Bool ok = false;
    while (!ok) {
        hw_sw_partition = codesign (
                                abstraction_level,
                                method_properties);
        ok = coverification (
                        hw_sw_partition,
                        design_constraints);
    }
    hw_sw_system = integration (
                            compilation_or_synthesis (
                                hw_sw_partition) );
}

```

Figure 3: Iterative hardware/software codesign

When data abstractions are used to represent hardware a class corresponds to a set of elements with common static and dynamic characteristics. Thus, a hardware component can be treated as a class containing state information along with a collection of associated operations that can manipulate this state. For example, a register can be viewed as a class with the operations read and write associated with it. The content of the register corresponds to its state, which can be accessed and manipulated using the operations read and write, respectively. Although combinational elements do not possess state, the concept of class can still be used to capture the common properties of these devices. For example, an arithmetic logic unit class may support the execution of different operations. At a higher level of abstraction, a processor contains a state, consisting of the program counter and other internal register values, which is manipulated by its supported instructions.

Software engineering uses data decomposition techniques [Jalote91] to derive implementations for abstract data types; this can be extended to hardware, modeled as data abstractions, as suggested in Figure 3. Before lowering the abstraction level for any used hierarchy, inheritance is applied: Starting with a collection of base classes,

it is possible to derive more specialized classes of components through inheritance. For example, the interface class can be used to derive intelligent interface classes; or specialized processors for the microsystem can be inherited from a general processor class.

Type genericity (a form of polymorphism) is the ability to parameterize with types a software element, such as procedure or data type. It makes programs more general. The template concept, that realizes type genericity in C++, can be extended to hardware components that act as containers.

## 6. DIGITAL-ANALOG SIMULATION

The essential difference between analog and digital simulation paradigm is induced by that between the mathematical structures their models are based on: algebraic - for digital, analytical - for analog. The discrete nature of simulation (design is a sequence of decisions, verification implies a sequence of stimuli) does not easily match the continuity of analog properties/parameters. Usually, the difficulty of analog simulation is avoided by defining an auxiliary representation domain, intermediate between the behavioral and the structural, where the problem is decomposed into topology selection and dimension computation. The first process is discrete and the second one is continuous over a restricted problem space. Object-oriented representation lends itself for this form-content complementarity application. But, topology selection would be more systematic if continuous modifications of the form were possible, and dimension computing is more efficient if symbolic algebraic methods are used.

The compromise between "simulation algebra" and "analog analysis" should be searched in three directions: 1) defining upper levels of abstraction for the analog domain, governed by algebraic laws; 2) modeling analog simulation in algebraic-analytical structures (functional analysis); 3) association of analytical syntax to the analog simulation process. All these approaches can be aided by an object-oriented Analog Hardware Description Language (AHDL) [Niculiu96].

## 7. THERMAL-ANALOG SIMULATION

The development of CAD procedures for microsystems imposes the simulation of thermal phenomena as secondary effects to the main, analog (electronic, mechanical, optical, and chemical), ones. As the microsystem components are modeled in AHDL the models can be enhanced with temperature dependence and power generation

estimation. Moreover models for environment and packaging conditions can be added as well. AHDL models permit direct simulation of the microscopic thermal transfer, and qualitative simulation-oriented representation of second order effects. Consequently, different physical domains, described by isomorphic analog laws, can be simulated in a unique representation. Dynamics, circuit theory, hydrodynamics, thermodynamics, electrostatics can be expressed with through-across concepts governed by dual topological laws for continuity and compatibility.

AHDL enables a direct physical simulation of heat conduction, alternative to discretization of the heat equation: only the first order relation representing Fourier's hypothesis is put in a AHDL model; its integration and discretization are realized by the topological constraints that characterize AHDL structures. This suggests the idea that we follow towards formal verification: Simulation is computer-oriented theory.

## 8. CONCLUSIONS

A unique representation for design and verification, supporting different hierarchy types, using object-orientation and constructive mathematics, permits intelligent simulation of complex systems containing parts of different physical, behavioral, or structural nature, e.g., intelligent interfaces of microsystems. In this paper we proposed an appropriate simulation framework architecture to support hierarchical description, representation, and simulation of the microsystem to be designed. We motivated also the introduction of the intelligent interface concept that have to cover and implicitly, to be represented by unifying models for, a number of complementary domains. Our proposal opens the way for formalizing representations in constructive mathematics style, and for multi-hierarchical simulation procedures.

## 9. REFERENCES

- Bibel W. et al., *Wissensrepräsentation und Inferenz*, Vieweg, 1993.
- Booch G., *Object-Oriented Analysis & Design*, Benjamin/Cummings, 1991.
- Jalote P., *An Integrated Approach to Software Engineering*, Springer, 1991.
- Niculiu T. et al., *O-O AHDL Models*, Electronic Design Automation Conf., Cambridge, p.291-292, 1996.
- Niculiu T. et al., *Hierarchical Simulation of  $m$ systems*, Technical Bulletin of University POLITEHNICA Bucuresti, EE, nr.1-2, p.47-58, 1998.
- Turner R., *Constructive Foundations for Functional*

*Languages*, McGraw Hill, 1991.  
Weiler C. et al., *C++ Base Classes for Hard/ Soft Co...*,  
Computer HDL Conf., Kyoto, p. 777-784, 1995.  
Winston P., *AI*, (3rd ed.) Addison-Wesley, 1992.