HIERARCHICAL INTELLIGENT MIXED SIMULATION

Tudor Niculiu
Bucharest University of Technology
Splaiul Independentei 313
77206 Bucuresti, Romania
tudor@messnet.pub.ro

Sorin Cotofana
Delft University of Technology
Mekelweg 4
2600 GA Delft, The Netherlands
S.D.Cotofana@dutepp0.et.tudelft.nl

KEYWORDS
Simulated Intelligence, Cosimulation, Hierarchy Types.

ABSTRACT

Intelligence is complementary to faith - common name for (intuition, inspiration, imagination) - and supposes, at least, {conscience, adaptability, intention). Conscience simulation demands transcending the present limits of computability, by an intensive effort on extensive research to integrate essential mathematical and physical knowledge guided by philosophical goals. A way to begin is hierarchical intelligent mixed simulation. Applying "Divide et Impera et Intellige" to hierarchy types reveals their comprehensive constructive importance based on structural approach, symbolic meaning, object-oriented representation. Formalizing hierarchical descriptions, we create a theoretical kernel that can be used for self-organizing systems.

INTRODUCTION

An algorithm is an entity that can be computer simulated, so it represents computability, bottom-up (construction, design, plan) or top-down (understanding, verification, learning). The algorithmic approach is equivalent to the formal one: If a sentence of a formal system is true, then an algorithm can confirm it. For a verification algorithm of the mathematical sentences a formal system can be defined, that holds for true the sentences in the set closure of the algorithm's results towards the operations of the considered logic. Formal systems, partial recursive functions, Turing machines, λ-calculus, Markov-algorithms, are only the best-known formalisms for computation, that is, also for algorithm and computability. Knowledge and construction hierarchies can cooperate to integrate design and verification into simulation; object-oriented concepts can be symbolized to handle data and operations formally; structural representation of behavior manages its realization. Hierarchy types open a way to simulate intelligence as an adaptable conscience.

HIERARCHY TYPES

Coexistent interdependent hierarchies structure the universe of models for complex systems. Abstraction and hierarchy are semantics and syntax of a fundamental concept, the most powerful tool in systematic knowledge: hierarchy results formalizing abstraction (Niculiu and Cotofana 2001). Hierarchy types reflect abstraction kinds. The abstraction goal is shown$\uparrow$:

- Class hierarchy ($\uparrow$concepts) $\leftrightarrow$ virtual framework to represent any kind of hierarchy, based on form-contents dichotomy, modularity, inheritance, polymorphism.
- Symbolization hierarchy ($\uparrow$mathematics) $\leftrightarrow$ stepwise formalism for all kind of types, e.g., for hierarchy types.
- Structure hierarchy ($\uparrow$managing) $\leftrightarrow$ stepwise managing of all (hierarchy) types on different levels by recursive autonomous block decomposition, following the principle "Divide et Impera et Intellige".
- Construction hierarchy ($\uparrow$simulation) $\leftrightarrow$ simulation framework for design/ verification of autonomous levels corresponding to different abstraction grades of description.
- Knowledge hierarchy ($\uparrow$theories) $\leftrightarrow$ reflexive abstraction ("in a deeper sense"), aiming that each level has knowledge of its inferior levels, including itself.

Understanding and construction have correspondent hierarchy types: their syntax relies on classes, the meaning on symbols, and their use on modules. The different hierarchy types can be formalized in the theory of categories. Constructive type theory permits formal specification and formal verification by generating an object satisfying the specification.

Knowledge-based architecture separates representation from reasoning. An intelligent system should be capable of reflexive abstraction. It 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.

.

HIERARCHICAL COSIMULATION

Simulation should remain correct, with increasing complexity, optimization and multi-domain requirements for the object-system. The hierarchical principle, applied to knowledge and simulation, (locally) bounds the complexity, by problem decomposition, and assures (almost) correct-by-construction design and efficient (design-adapted) verification.

Hardware - Software Cosimulation

The hardware-software cosimulation of complex systems is imposed by the incompatibility/ 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, can be used for hardware.

Knowing the features (mandatory: abstraction, hierarchy, encapsulation, modularity, message passing + optionally: typing, concurrence, persistence) that characterize an object-oriented language, they also make sense from the perspective of hardware modeling and simulation. Object-oriented specification of models can be based on general systems theory, what makes this approach applicable in all domains. The designed framework permits self-organizing. It offers at any abstraction level of the simulation hierarchy: system description in a commonly used language extended for parallelism by synchronization items; automatic learning-based hardware/ software partition of the description; consistent communication between heterogeneous parts and with the exterior; simulation of the whole system during any design phase.

Data abstraction can be 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 class containing state 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. The contents of the register correspond to its state, which can be accessed and manipulated using the operations read and write, respectively.

Software engineering utilizes data decomposition to refine (derive implementations for) abstract data types. When modeled as data abstractions, hardware elements can also be refined using this decomposition technique. Generic types (a form of polymorphism) result from the ability to parameterize with types a software element, such as procedure or data type. This makes programs more general. The template concept, that realizes it in C++, can be applied to hardware components that act as containers, e.g., registers, register files.

Digital-Analog Cosimulation

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. In view of intelligent simulation the whole intelligence has to be simulated, i.e., conscience along with adaptability. The discrete parts of simulation, e.g., a sequence of decisions/ stimuli for design/ verification, do not easily match the continuity of analog properties (Blum et al. 1998).

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 instance. 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.

We searched the compromise between simulation algebra and analog analysis 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 (of 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).

Thermal-Analog Simulation

The development of CAD procedures for micro-systems imposes the simulation of thermal phenomena as secondary effects to the main, analog (electronic, mechanical, optical, chemical), ones. As the micro-system 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, electrodynamics 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 discrete heat equation: only the first order relation representing Fourier's hypothesis is expressed in an AHDL model; its integration and discretization are realized by topological constraints that characterize AHDL structures. This suggests the idea that we follow towards formal verification: Simulation is computer-oriented theory.

Behavioral Adaptable Design for Testability

Design-for-testability (DFT) must suit the behavioral specification of today's complex system design Referring to high-level synthesis, DFT can operate before, while or after it. The first choice permits the intervention of an intelligent agent for adapting the DFT technique, model or method to the particular design. We call it behavioral adaptable design-for-testability: it improves the testability, measured with adequate methods, direct on the behavioral specification or aided by special representations, that have to permit returning to the behavioral description after improving the testability of the system to be designed. The results are general enough to be valid for systems, either hard, soft or hard/soft.

Memory elements - registers (arrayed flip-flops)/ flip-flops/ latches (unclocked flip-flops) - are represented in behavioral hardware descriptions by variables or signals. Variables are description objects local to processes/ subprograms, used to store intermediate values between sequential statements, characterized by free assignment. Signals are permanent description objects to link concurrent elements: components/ processes/ concurrent assignments, demanding synchronized assignment, declared locally - within architecture, block or other declarative region, or globally - in extended package. In the context of a process synchronized by a clock signal, in a behavioral description, signals implicated in simple/ multiple signal assignment generate memory during synthesis. An analog rule can be formulated for variables: Inside a process, a variable that must hold values between iterations of the process implies memory elements. A variable that is set but not used between synchronization statements infers memory; a variable that is read before being assigned also infers memory. The context is not restrictive, as all concurrent statements are equivalent to processes (excepting direct instantiation). For called subprograms the rules of memory inference can be deduced directly: pure functions (no side effects) do not - while procedures (side effects) do infer memory elements.

The most used DFT techniques are Scanning, Built-In Self-Test and Test Point Insertion. They can be applied at the different levels of the design hierarchy (behavior, RTL, logic) and can be combined. We began with Partial Scan applied to the autonomous blocks of the behavioral HDL specification, but the other techniques can contribute to improve the testability of the behavioral specification or the way to this goal. All types of hierarchies are implied in this approach: design abstraction levels, block structure, class/ object framework, symbolization and knowledge hierarchies. The Partial Scan problem is the selection of the scan registers following a strategy to find an optimum testability - complexity compromise. We combined Partial Scan methods to optimize the order to add memory elements to the scan chain, at behavioral level. An adaptable interface assures the translation, in both senses, from behavioral hard/ soft description to a structural representation of the required behavior. The partial-scan selection uses a knowledge base to generate the weighted directed graph (flip-flops, combinational paths) and to return to text the differences caused by transformation for testability improvement. The rules of correspondence between description object (signal/ variable) assignments and registers, and those to translate the data flow in the behavioral specification to weighted arcs in the graph counterpart and to combine different testability measures in node weights, guide the first step. The second step is solved by incrementing rules for the hard/ soft description. Partial-scan needs for the return translation a pointing scheme for the scanned objects among signals/ variables of the behavioral specification. This is managed by an adequate data structure in HDL. In principle, flip-flops are selected for scan, but when a register is used parallelly, it is candidate entirely for scan. The variables/ signals inferring memory are testability-related sorted to select incrementally the scan elements that will be eventually mapped to the scan register.

INTELLIGENT SIMULATION

Faith and intelligence are yin-yang of our life (Way, Truth, Life).

- Faith = (Inspiration, Intuition, Imagination) is associated to the right human brain hemisphere. Intuition is the main part of the dark yin, inspiration the dynamical shaped interface to intelligence, the white point link stands for imagination.
- Intelligence = (Conscience, Adaptability, Intention) is linked to the left human brain hemisphere. Adaptability is the main part of the light yang, conscience the variable interface to faith, the dark point, sent by faith, signifies intention.

Conscience is self-awareness of individual faith and intelligence, as well as of the relation to the local context (society) and to the global one (universe). To appear it needs self-knowledge, what could result from community conscience featured by an eternal human structure, e.g., from the past, shepherds, farmers, sailors, Africans, American Indians, Asians, Australians, ... Each individual recognized himself in his cohabitants, besides being adaptable and having a lot of intuition. The alternative eternity/ evolution can be explained by the kind conscience sees the others intelligent agents: as similar, respectively, as different to itself. The evolution of the common measure is conditioned by the conscient construction of communicating individual intelligence to manage the lower stages, as industry enabled the mechanization of agriculture followed by the concentration on economics.

Evolution implied a multiple "Divide et Impera et Intellige" for conscience, associated to generating the *components* lacking of the mind at start, then assisted by them:

- individual-social-universal conscience$\rightarrow$*inspiration*
  (subjective-contextual-objective) $\downarrow$
- space-time (structure-behavior) $\rightarrow$ *imagination* $\downarrow$
- discrete-continuous (natural-real) $\rightarrow$ *intention*
- beauty-truth-good (art-science-technology)¯.

The convergence process of evolution demands struggle against time, with structure as ally. Conscience needs perhaps, more than discrete recurrence, continuous feedback. Social and individual conscience are mostly divergent nowadays, i.e., we only performed "Divide et Impera", neglecting "et Intellige". It's high time to correct this.

SEARCHING FOR CONSCIENCE

The anterior relations are oversimplified in order to move towards intelligent simulation. Although we claim they are intuitive and hope they are inspired, to begin, we neglect the essential but too primitive to understand intuition and inspiration, so formalizing reflexive abstraction by the knowledge hierarchy type and simplifying abstraction mainly by the simulation hierarchy type, it follows that:

$$\text{Conscience} = \text{knowledge}(\text{simulation}(\text{Conscience})) \tag{1}$$

i.e., a fixed-point relation suggesting that we could model conscience associating to any hierarchical level of the construction process a knowledge level. The fixed-point problem needs a metric space where knowledge $\circ$ construction is a contraction, i.e., elements implied in the construction should get closer to one another in the formal understanding of the formal construct. If, even in the sketch, we consider general functional relations between the essential parts of the faith-assisted intelligence:

$$\begin{aligned}\text{Conscience} = \text{knowledge}\ (&\text{intention}\ (\text{Inspiration}, \\ &\quad \text{simulation}\ (\text{imagination}\ (\\ &\qquad \text{Intuition}, \\ &\qquad \text{Conscience}) \\ &\quad) \\ &) \\ )\end{aligned} \tag{2}$$

RESEARCHING FOR CONSCIENCE

Alternative ways to extend the computability concept can be compared to universal approaches known from German novels:

1. Faust (Johann Wolfgang von Goethe): heuristics - risking competence for performance, basing on imagination, confined to the mental world.
2. Das Glasperlenspiel (Hermann Hesse): unlimited natural parallelism - remaining at countable physical suggestions, so in the Nature.
3. Der Zauberberg (Thomas Mann): hierarchical self-referential knowledge - needing to conciliate the discrete structure of hierarchy with the continuous reaction, hoping to open the way to Reality.

They concentrate respectively on the mental world of the good managed by technology, the physical world of the truth researched by science and Plato's world of the beautiful abstractions discovered by art (Amoroso et al. 2000).

Recurrence is confined to discrete worlds, while abstraction is not. This difference suggests searching for understanding based on mathematical structures that order algebra into topology (Rozenberg and Salomaa 1993). We follow the (mathematical) paradigm of intelligent simulation, by functionally modeling the self-aware adaptable behavior for intelligence simulation. The integration between discrete and analog is again needed, for a soft adaptability and for conscience simulation as continuous reaction (Traub 1999).

REFLEXIVE KNOWLEDGE

Mathematics contains structures suggesting to be used for self-referent models. The richest domain for this is functional analysis, which integrates algebra, topology and order: contractions and fixed points in metric spaces, reflexive normed vector spaces, inductive limits of locally convex spaces, self-adjoint operators of Hilbert spaces, inversable operators in Banach algebra. A hierarchical formal system can be defined:

1. $(U, \{H_i \in S_h), \mathrm{card}(U) > \aleph_0$ // hierarchical universe
2. $\Sigma = F \cup L \cup A \cup K$ // functional objects
   $F = \{f \mid f : U^* \rightarrow U\}$ // global functions
   $L = \{f \mid f : Level_j^* \rightarrow Level_j\}$ // level structures (3)
   $A = \{f \mid f : Level_j^* \rightarrow Level_{j+1}\}$ // abstractions (- K)
   $K = \{f \mid f : Level_j^* \times Level_{j+1} \rightarrow Level_{j+1}\}$ // knowledge
3. $I = \Sigma^* \cap R$ // initial functions
4. $R = \{r \mid r : \Sigma^* \times R^* \rightarrow \Sigma \times R\}$ // transformation rules.

U is a category, e.g., containing Hilbert spaces with almost everywhere-continuous functions as morphisms, enabling different ways to simulate self-awareness (Ageron 2000). For example, considering self-adjoint operators as higher-level objects of the knowledge hierarchy, these levels can approach self-knowledge like knowledge about the inferior levels and of the current one, having also some qualitative knowing about the superior levels. The correspondence problem, to associate the knowledge hierarchy to the simulation hierarchy, is managed by natural transformations over the various functors of the different hierarchies regarding the simulated system.

To complete the simulation of the intelligence's components, intention is first determined by human-system dialogue (Keutzer et al 2000). Further than modeling conscience to simulate intelligence there will be searching to comprehend inspiration, may be using Lebesgue measure on differentiable manifolds or non-separable Hilbert spaces. To approach intuition even mathematics has to be more philosophy-oriented.

CONCLUSIONS

Simulation is algorithmic theory. Cosimulation needs consistent combination of mathematical domains, as well as an intelligent compromise between consistence and completeness. Intelligence simulation implies a hierarchical approach of different types. Knowledge hierarchies demand for extension of the algorithm concept. Searching for different computer architectures, various combinations of hierarchies, i.e., of abstractions of various kinds, several mathematical ways to extend the computability domain are to be followed.

Simulability is computability using the power of continuum. Positive signs to confirm this come from analog electronics, control systems, mechatronics. Real progress towards this new way of computation needs unrestricted mathematics, integrated physics and metaphorical thinking. Evolution needs separation of faith and intelligence, understanding and using consciously more of faith's domain, integrating them to human wisdom, to be divided further to get more human. Metaphorically said, the current problem is: *God's ways are uncountable*.

REFERENCES

Ageron, P. 2001. "Limites inductives dans les categories accessibles". In *Theory and Applications of Categories*, 8, No.11, 313-323.

Amoroso, R., R.Antunes, C.Coelho, M.Farias, A.Leite and P.Soareset (Eds.). 2000. *Science & the Primacy of Consciousness*, Noetic Press.

Blum, L., F.Cucker, M.Shub and S.Smale. 1998. *Complexity and Real Computation*, Springer Verlag.

Keutzer. K., S.Malik, A.R.Newton, J.M.Rabaey and A.Sangiovanni-Vincentelli. 2000. "System-Level Design: Orthogonalization of Concerns & Platform-based Design". In *IEEE Transactions on CAD of Integrated Circuits and Systems*, 19, No.12, 1523-1543.

Niculiu, T. and S. Cotofana. 2001. "Hierarchical Intelligent Simulation". In *Proceedings of the European Simulation Multiconference* (Prague, June 6-9). A Publication of the Society for Computer Simulation International, 243-246.

Rozenberg, G. and A.Salomaa (Eds.). 1994. *Developments in Language Theory*, World Scientific.

Traub, J.F. 1999. "A Continuous Model of Computation". In *PhysicsToday*, May, 39-43.

AUTHOR BIOGRAPHY

Tudor NICULIU: Assoc.Prof. EE Dept., "Politehnica" University of Bucharest. He is looking for hierarchical integration of different domains, as hard & soft, discrete & continuous, electrical & unelectrical in order to simulate intelligence for intelligent simulation.

Sorin COTOFANA: Assoc.Prof. EE Dept., Delft University of Technology. His research interests include computer arithmetic, custom computing machines, embedded systems, logic design, parallel architectures, neural networks, computational geometry, and CAD.