Validation and Verification of Timing Properties of an Architecturally Described System.

Laurent Thomas

THOMSON-CSF Corporate Research Laboratory
Domaine de Corbeville, 91404 ORSAY Cedex, France
Tel: (+33) (0)1 69 33 92 92
Fax: (+33) (0)1 69 33 08 65
Email: thomas@thomson-lcr.fr

Abstract

We are investigating how formal methods can help in establishing global properties of a system's model from the knowledge of properties of its components. It appears that no known model of compositionality yet fully satisfies properties preservation. A general framework for verification and simulation is needed in order to assist in the early phases of design. We strongly believe that the **executable** architecture description is such a framework. Properties verification, and especially timing properties verification, is of paramount importance in the correction of the systems developped by THOMSON-CSF, and is thus our primarly goal. Hybrid system modelling permits the description of components with emphasis on timing properties. However the level of description of the cooperation of such hybrid components is rather primitive. Thus, formal definitions and properties of various compositions schemas of hybrid components have to be investigated.

Keywords: Hybrid components, Validation, Verification, Architecture, Mathematical models of composition.

Workshop Goals: Networking, Brainstorming

1 Background

It is now a commonplace idea to stress the fact that industry needs tools to help in the design of (correct) computing systems. Reuse, simulation and testing have been advocated as good[1] industrial practices that should help in devising correct[2] products. However, recent (computer caused) disasters such as the Ariane 5 first flight [LeL96] illustrate too well how such a “proven” component can badly fail when inconsiderately reused.

Being a world leader in electronics products, THOMSON-CSF has engaged through its Corporate Research Laboratory a wide scale research activity in the architecture and system technologies. The long term goal is to provide tools to designers to help them design systems and verify their properties before producing a single line of code. The derivation of timing constraints and the construction of a system structure that guarantees to satisfy those constraints are the core objectives of our ongoing research.

In fact, modelling a system can be sliced in two parts [Rus97]: requirements elicitation and requirements engineering. As noted in [Rus97]:

> Requirements elicitation is concerned with discovering what is wanted; [...] Requirements engineering, on the other hand, is concerned with turning the products of requirements elicitation into precise, unambiguous, and complete description of what the system under consideration is to do.

Formal methods, because they support analysis through formal deduction, are an appropriate technique in requirements engineering : its mathematical modelling and calculation oriented capability allow consequences of requirements and properties of the design to be predicted and evaluated before a line of code is written.

Our (partial) solution involves the **executable** description of the system architecture as being the model of the future system. A system’s software architecture [GS93] and its description in an architectural description language [Ves93, Med97] are the common framework shared by clients, users, architects and developpers. A system’s architecture is a set of dynamically connected dynamically created and destroyed components. Properties of the overall system are usually derived through the execution of its architecture. We aim at introducing more “formality” in this design process. Recent developments in formal methods [Rus96] convinced us that a formal approach to the certification of, at least the critical points, is now feasible. Thanks to the architectural approach, the designer of the (model of the) system has to reason about components and component interconnections. Components’ formal specification process will force the designer to elaborate very precisely the behavior of each component, exposing ambiguities early in the design process. Formal proofs on the behaviour of composed components and its architecture execution will give the users and designers confidence in the system.

It remains to understand how formal methods and composition of well described components can harmoniously fit together in order to help ensure properties on the composed model. For instance, timing constraints are usually global constraints (time being a global variable). It is not clear whether and how they can be obtained compositionally.

[1] read : the only feasible
[2] read : seem to give the predicted answers

2 Position

Industry, and especially high technology industry, relies on a set of developed products to accomplish a given task. It is common practice to reuse already developed components, whether "home made" or off the shelves. Those components have to be composed, in some way, and there are many of them, to become the system's model.

The need to know, before completion (and even more before begining the developpement itself), that a product will match the clients requirements is of paramount importance. It is thus crucial to know in advance (before begining the development phase) that a system can be constructed that will respect certain properties, knowing the components and their properties it will include.

The standard (mathematical) approach to verification requires a *model* of the system being verified, a *specification method* for expressing desired properties and a *proof method* to show that a model satisfies those properties.

The problem of composition is the problem of the non closure of properties. For instance consider the nonzenoness property (every finte run can be extended to an infinite one) : let P be a component emitting a message at time 0, and upon receiving a message at time δ_i, emits at time $\delta_i + 1/2^i$ for $i > 0$. Now let Q be another component waiting for an input and once the input arrives at time σ_i, it emits a message at time $\sigma_i + 1/2^i$, for $i > 0$. Each component is nonzeno, but their parallel composition is not (time cannot progress past 2).

Figure 1 illustrates that point : what are the "good" abstraction functions and composition operators such that the diagram commutes and such that there exists a way to conclude about the validity of a property on C from those known about $\mathcal{M}_A$ and $\mathcal{M}_B$?

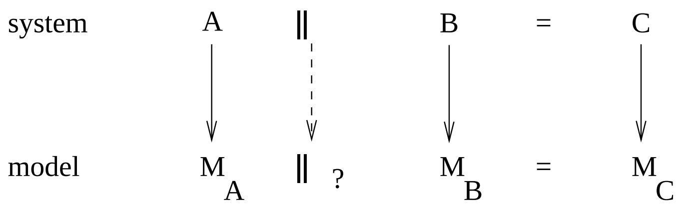


Figure 1: The composition problem

2.1 Composition schemas

To deal with composition, we first need to understand how to describe components' behaviours, and then investigate how the different formalisms may be composed to describe the behaviour of the full system. For instance, [BS97] insists that not only should components specify what they do but also the minimal contracts their required components must obey.

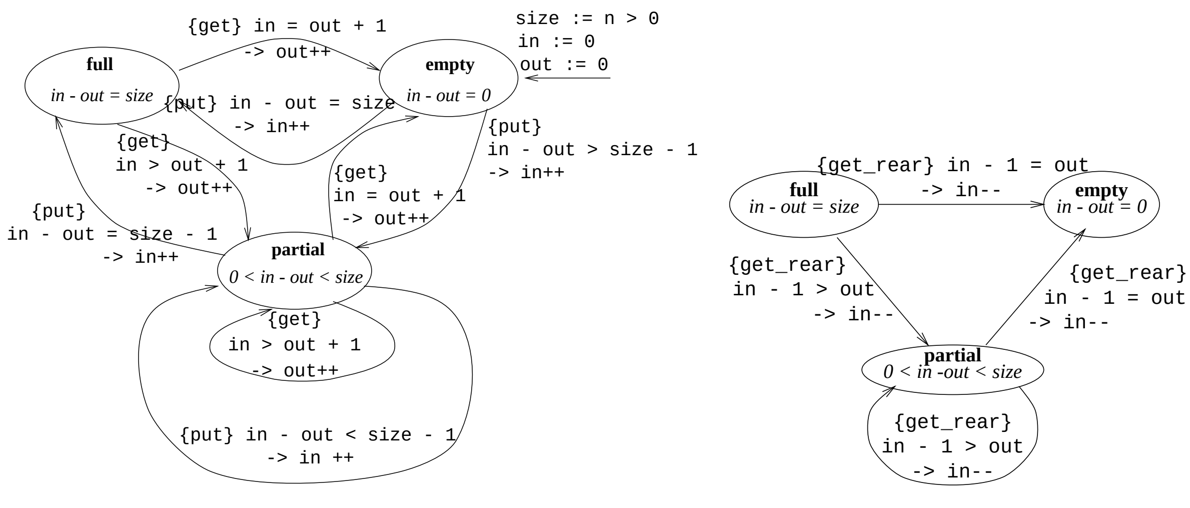
Behaviours can be specified in at least two main ways : automaton or theory (logic). Properties can be specified through logic, whether modal or not, or in a form of an automaton.

In this position paper we will only cite a small subset of those ways.

2.1.1 An example of composition formalism

A guarded automaton is a way for describing the behavior of a component's internal state and set of operations acting on the component's state. Transitions describe how component's internal state changes through activation of the component's permitted operations.

More precisely, a guarded automaton is a finite state automaton whose states and edges are labeled by respectively an automaton state invariant and a tuple (o, g, a), where o is the name of an operation, g a guard (a predicate on the component internal state), and a an action (a change in the component's internal state). The intended semantics is as follow: an (automaton) state σ refers to a set of component's internal valuations. The (automaton) state's invariant has to be true in all component states refered by this (automaton) state. Being in the (automaton) state σ, a transition to τ can be fired if there is an edge from σ to τ labeled (o, g, a) such that the next to be done operation is o and the guard g evaluates to true. If the transition is fired, the action a changes the (model) states. The (automaton) state invariant in τ must then be verified by the new (model) state. For instance, consider a (circular) bounded buffer whose operations are `get` and `put`. It can be modeled as a three states automaton whose states are *full* (intended meaning : no element can be inserted), *empty* (intended meaning : no element can be retrieved) and *partial* (intended meaning : elements can be inserted and retrieved). Two indices, *in* and *out* respectively point to the element's insertion and retrieve buffer's location. Figure 2, on the left, shows the corresponding guarded automaton.



(a) *Bounded Buffer* Guarded Automaton (b) *Get Rear* Guarded Automaton

Figure 2: Guarded Automata of a circular bounded buffer *(see 2.1.1)*.

Suppose we want to create a component that extends the previous bounded buffer with a `get_rear` operation (an operation retrieving the last insterted element). The guarded automaton at the right of figure 2 presents such an operation. The behavior of the extended bounded buffer can be described as a (guarded automaton) product where guards and invariants are ANDed.

But a more complicated example, where automata states are distincts, will require a more complicated set of composition rules. It may also appear that, for some properties, ORing, rather that ANDing, guards and invariants will be more appropriate.

2.1.2 Introducing Time

Two kinds of properties are of particular interest in the verification of systems, namely, *safety* and *liveness.* Temporal Logic [Eme90] helps in defining and verifiying properties on the sequence of timed events, that is their relative order in time of their occurrence. Composition reasonning shemas like compositional minimization and assume/guarentee have been investiguated [CLM89, GL94] but seem not to have received enough attention to be at the point of being transferrable into industry. Moreover, timing constraints being our main topic rather the pure partial order of events, we turned to the timed automata extensions we will speak about now.

2.1.3 Valuating Time

Precise timing valuation are needed in critical systems. Modelisation decisions like *linear* versus *branching* time or *discrete* versus *dense* time, point in time logic versus interval logic lead to different kinds of provable properties.

A (model of a) system may be composed by components whose time modelisations and synchrony assumptions are different. Their level of description may also differ. Verification techniques for those systems should thus be scalable along time and space axis [AH96].

Among properties of particular intererest to us, the tuning of components is one of the most important. Components off the shelves must usually be tuned/adapted before being integrated in a system [BBRR97, BR97].

As a scholarly example of tuning let's consider the classical Fischer's real-time mutual exclusion protocol for two processes P_1 and P_2 :

P_i :
  repeat
    repeat
      await k = 0
      k := i; **delay** b
    until k = i
    CS_i
    k := 0
  forever

Variable k is shared between the processes and is initially set to 0. Exclusive access to their respective critical section then relies on two parameters : the delay b and the time a it takes a process to write a value in a variable. To complicate matters, the processes use drifting clocks, P_1's being slow and P_2's being fast.

Figure 3 presents the behavior of the P_i's in the hybrid automaton formalism [AD94, Hen96]. The clocks' speeds are x and y.

HyTech [HHW97] verification's algorithm on the automata of figure 3 gives the condition $8b > 11a$ for the safety property P_1 *and* P_2 *are never in their critical section at the same time* to be true.

Formal descriptions should be modular. However correct modules/components can be connected so that the overall behavior is no more correct as illustrated by the following example.

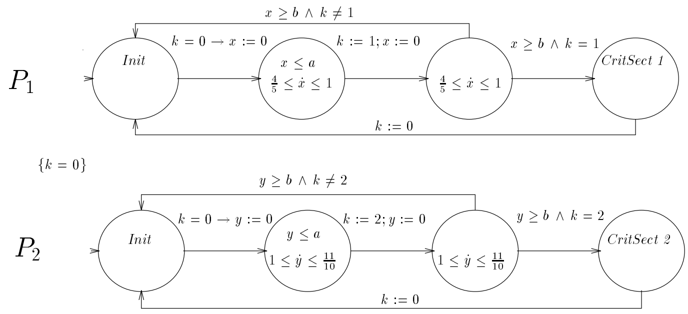


Figure 3: Fischer's mutual exclusion protocol

This example is taken from [SY96]. A medium receives messages from a sender and transmits them to a receiver. Two consecutive inputs are separated by 7 units of time, two consecutive outputs are separated by at least 4 units of time, and the transmission delay between an input and an output in is the interval $[3, 8]$. See figure 4. If the delay interval is replaced with $]7, 8]$, a deadlock occurs in ABD with clocks x and z equal to 7 : time cannot progress and `out` is not enabled.

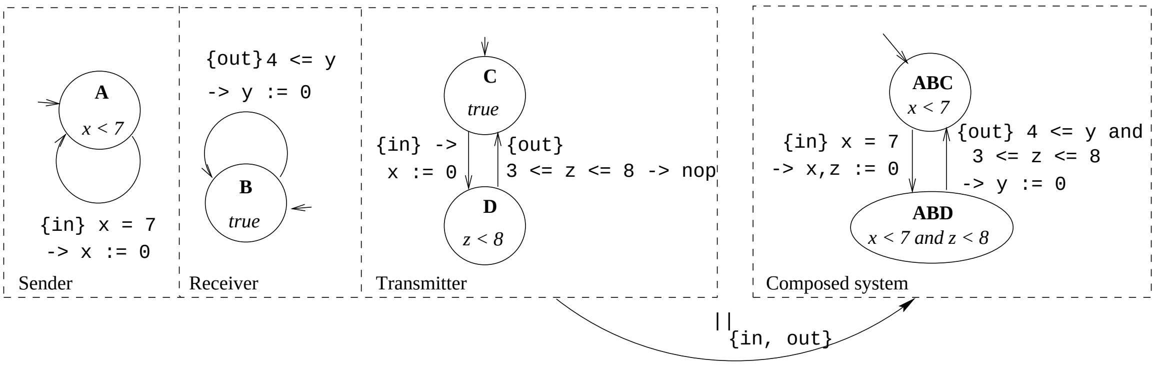


Figure 4: A channel modelisation [SY96]

Fortunately some techniques exist to detect if an hybrid automaton is non Zeno [HNSY94].

Composition schemas of hybrid automata rely heavily on when transitions of the composed automaton will be permitted to fire. Let $[l_A, u_A]$ and $[l_B, u_B]$ be the bounds associated with two transitions of automata A and B. The bounds on the corresponding transition of the composed automaton could be $[\max\{l_A, l_B\}, \min\{u_A, u_B\}]$ or $[\max\{l_A, l_B\}, \max\{u_A, u_B\}]$ or $[\min\{l_A, l_B\}, \min\{u_A, u_B\}]$. Depending on the choice, properties may or may not be verified on the composed system. It remains to understand the semantics of such composition schemas. As noted in [SY96], when composing timed systems, three independent parameters operate :

① the parallel composition for the underlying untimed system,

② the composition rules for guards,

③ the composition rules for time constraints

Can we exhibit rules for choosing a particular composition schema ?

A related question is : "can we exhibit a sufficiently powerful execution model to describe behaviors ?". For instance the *partially ordered multiset* of events execution model of Pratt

[Pra86] is used in the executable architectural description language Rapide [LKA+95] and many applications were claimed to be simulated with it. However we were not able to use this model to describe the famous train-gate controller problem [HL94], due to the inability of the model to test and discard time valued events.

Another question has to do with the various formalisms that can be used to describe independantly developped components. How can those formalisms interoperate in a common framework when they are integrated in the complete application's model ? Synchronous languages like Esterel [BG92] and Lustre [HCRP91], despite having completly different execution models, rely on a common intermediate formalism, which permits components developped in those two different languages to interoperate at the level of their execution.

2.2 Proposition

> Many of the problems in software and hardware design are due to imprecision, ambiguity, incompletness, misunderstanding, and just plain mistakes in the statement of top-level requirements, in the description of intermediate designs, or in the specifications of components and interfaces. [Rus92]

Our current approach to the modelisation and verification problem is to define a system by its architecture. An architecture reflects the structure of a system in term of a set of its (dynamically created and destroyed) components, the nature of the connections between them, and its (dynamically evolving) topology. Each component can be described by many views depending on the peculiar properties we want to prove on their composition. Among those views, the composition rules, the behavior description formalisms and the kinds of properties to be verified are obvious choices.

Extending towards an architectural description language a programming language like Shift [DGS97] seems a better approach to make complex systems' designers confident in the possibility of specifying and verifying properties before implementation.

Our proposal is to investigate how architecture description languages can help in the design and specification of various composition operators and help in the verification process of the overall correctness.

References

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–236, 1994.

[AH96] R. Alur and T. Henzinger. Reactive modules. In 11^{th} *IEEE Symposium on Logic in Computer Science*, pages 207–218, 1996.

[BBRR97] F. Boniol, A. Burgueno, O. Roux, and V. Rusu. Analysis of slope-parametric hybrid automata. In *International Workshop on Real-Time and Hybrid Systems, HART'97*, Grenoble, France, March 1997.

[BG92] G. Berry and G. Gonthier. The Esterel Synchronous Programming Language : Design, Semantics, Implementation. *Science of Computer Programming*, 19(2):87–152, 1992.

[BR97] A. Burgueno and V. Rusu. Task-system analysis using slope-parametric hybrid automata. In *Workshop on Real-Time Systems and Constraints, Euro-Par'97*, Passau, Germany, August 1997.

[BS97] M. Büchi and E. Sekerinski. Formal methods for component software: The refinment calculus perspective. In *Second International Workshop on Component-Oriented Programming*, 1997.

[CLM89] E.M. Clarke, D.E. Long, and K.L. McMillan. Compositional model checking. In *Proc. of the Fourth Annual IEEE Symposium on Logic in Computer Science*, 1989.

[DGS97] A. Deshpande, A. Göllü, and L. Semenzato. The SHIFT programming language and run-time system for dynamic networks of hybrid automata. UCB-ITS-PRR 97-7, California PATH Project, 1997.

[Eme90] E.A. Emerson. *Temporal and Modal Logic*, volume B. Formal Models and Semantics of *Handbook of Theoretical Science*, chapter 16, pages 995–1072. The MIT Press/Elsevier, 1990.

[GL94] O. Grumberg and D.E. Long. Model checking and modular verification. *ACM TOPLAS*, 16(3):843–874, May 1994.

[GS93] D. Garlan and M. Shaw. An introduction to software architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, volume I. World Scientific Publishing Compagny, 1993.

[HCRP91] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programmming language LUSTRE. In *Proc. of the IEEE*, volume 79(9), pages 1305–1320, September 1991.

[Hen96] T.A. Henzinger. The theory of hybrid automata. In *11th Annual IEEE Symposium on Logics in Computer Science (LICS 1996)*, pages 278–292, 1996.

[HHW97] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. In *Ninth International Conference on Computer-aided Verification (CAV 1997)*, Lecture Notes in Computer Science. Springer-Verlag, 1997.

[HL94] C.L. Heitmeyer and N. Lynch. The generalized railroad crossing: A case study in formal verification of real-time systems. Technical Report 7619, Naval Research Laboratory, Décembre 1994.

[HNSY94] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Inf. and Comput.*, 111(2):193–244, 1994.

[LeL96] G. LeLann. The Ariane 5 Flight 501 Failure – A Case Study in System Engineering for Computing Systems. Technical Report 3079, INRIA, December 1996.

[LKA+95] D.C. Luckham, J.J. Kenney, L.M. Augustin, J. Vera, D. Bryan, and W. Mann. Specification and analysis of system architecture using rapide. *IEEE Trans. on Soft. Eng.*, pages 336–355, April 1995.

[Med97] N. Medvidovic. A classification and comparison framework for software architecture description languages. UCI-ICS 97-02, Department of Information and Computer Science. University of California, Irvine, February 1997.

[Pra86] V. Pratt. Modelling concurrency with partial orders. *Int. Journal of Parallel Programming*, 1986.

[Rus92] J. Rushby. Formal methods for dependable real-time systems. In CNES, editor, *International Symposium on Real-Time Embedded Processing for Space Applications*, pages 355–366, Les Saintes-Maries-de-la-Mer, France, November 1992. Cepadues-Editions.

[Rus96] John Rushby. Mechanized formal methods: Progress and prospects. In *16th Conference on the Foundations of Software Technology and Theoretical Computer Science*, volume 1180 of *Lecture Notes in Computer Science*, pages 43–51, Hyderabad, India, December 1996. Springer-Verlag.

[Rus97] John Rushby. Calculating with requirements. In *3rd IEEE International Symposium on Requirements Engineering*, pages 144–146, Annapolis, MD, January 1997. IEEE Computer Society.

[SY96] J. Sifakis and S. Yovine. Compositional specification of timed systems. In *13th Annual Symp. on Theoretical Aspects of Computer Science*, volume 1046 of *Lecture Notes in Computer Science*, pages 347–359, February 1996.

[Ves93] S. Vestal. A cursory overview and comparison of four architecture description languages. Technical report, Honeywell Technology Center, February 1993.

3 Biography

Laurent Thomas is a Research Scientist at the Corporate Research Laboratory of THOMSON-CSF. He is leading a 3 year project on construction and validation of distributed real-time systems, funded by both the French Ministry of Research and the French Ministry of Defense. He received a Ph.D. in Computer Science from University of Nancy, France in 1992, and he spent 2 years at the University of Tokyo in the laboratory of Prof. Yonezawa as a European Post-Doctoral Fellow.

Acknowledgement: This research is partly funded by the French Ministry of Defense under grant 96.34.448.00.470.75.65 and the French Ministry of Research under grant 96C0076.