An incremental formal approach to real-time systems

Ana Férnandez Vilas and José J. Pazos Arias [1]
Área de Ingeniería Telemática. University of Vigo. 36200 Vigo. Spain
tel: +34 986 812186, fax: +34 986 812116.
{avilas, jose}@ait.uvigo.es

ABSTRACT

A real-time system is one in which the correctness of the system depends not only on the logical results, but also on the time at which results are produced. Formal approach to real-time systems has been tackled extending a large amount of untimed formalisms studied at length. Once major timed problems have been completely or partially resolved, our aim is slightly different. In order to improve software quality, our intend is to merge two different nature solutions: the formalization of the software process; and a process approach which is both iterative and incremental over the whole life cycle. In this way we combine the correctness of formal methods, and the suitability of a life cycle that follows the user requirements and splits complexity.

1 Introduction

A real-time system is one in which the correctness of the system depends not only on the logical results, but also on the time at which results are produced. In this field, there can not be any doubt that real-time formal methods have been delayed with respect to conventional ones, going back few years. Once maturity in formalization of untimed systems has been reached, formal approach to real-time systems has been tackled extending a large amount of formalisms studied at length.

Shortly after theoretical results appeared, verification tools were developed mainly in academic area (KRONOS followed by UPPAAL, RT-SPIN and Timed-COSPAN). All these tools are based on *timed automata* to a large extent, although they differ in their property specification languages. Real case studies (like the ones tackled by UPPAAL and KRONOS) have highlighted timed models and methods are already ripe for practitioners.

Nevertheless, these timed theoretical foundations come up against industry reluctance about formal methods adoption in the software process; technology-transfer problems, which have been discovered in conventional untimed systems, appear once again. We believe one key factor is unsuitable user-orientation of most academic FDT's (**F**ormal **D**escription **T**echnique) tools. It is urgent to improve helper mechanisms to make tools easier to use to professionals without strong theoretical background. Apart from aided issues, integrated environments are also essential to bring real-time formal methods closer to the user. It is advisable to include facilities ranging from static analysis, based on formal notations and schedulability theory, to dynamic analysis based on testing or run-time monitoring [1, 2].

With respect to software process, requirements for large and complex systems are always problematic initially and they evolve continually throughout the life cycle. So, software process models have to be robust and flexible in order to accommodate the inevitable and often continuous stream of changes. What is more, in early phases designer has not a deep knowledge of the system, and in any case, maybe formal complexity is excessive for one-step design. Traditional cascade life cycles are unlikely suitable. Therefore, formal methods should be adapted to support evolution, outside their traditional role of verifying that a model meets certain fixed requirements. See [3] for real-time scenarios where system evolution support is needed.

Once major timed problems have been completely or partially resolved, our aim is slightly different. In order to improve software quality, our intend is to merge two different nature solutions: the **formalization** of the software process (gaining the advantages of FDT's); and a process approach which is both **iterative and incremental** (fitting in with requirements change) over the whole life cycle. In this way we combine the correctness of formal methods, and the suitability of a life cycle that follows user requirements and splits complexity. This doctoral proposal is a timed extension of previous works tackled by my research group in the field of untimed reactive and distributed systems [4, 5, 6]. Main contributions in these works are the following: causation in temporal logic; incomplete specifications; suggestions computation; and translation from logic into FSM's (**F**inite **S**tate **M**achine) and from FSM's into LOTOS [7].

Theoretical works related to formal design and analysis of real-time systems (summarized in section 2) are the required elements to define a formal model of software process. But, given that the life cycle will be iterative and incremental, to reach a sound design, relating subsequent cycles to each other and recording design information in previous cycles is needed. In every cycle, the system capability to satisfy user requirements depends on previous requirements enforced in the system. Once a formal design is obtained, it is a good idea to use this design for several kinds of analysis (formal, and even informal).

[1] Advisor of the doctoral work introduced

Finally, although the primary aim of my doctoral work is not focused on efficiency issues related to formal analysis, we try to depict the state of the art in this field (mainly by minimization).

2 State of the art

In this section we summarize theoretical results in real-time models, logics and methods. In order to formalize a software process, integrating the best practices in the state of the art is needed.

One primary question is to select a realistic model for the time. Although there has been a long and still unsettled debate concerning the selection of a discrete or a dense time model, there is a broad consensus [8, 9, 10] that dense models are more expressive and suitable for composition and refinement. Given that algorithms over dense-time models are increasingly efficient every day, we consider dense time models are more correct and natural since, moreover, there are application areas, like hybrid systems, where time can not be discretized fine enough.

Timed formalisms embrace several approaches which differ on their methods, aims, and abstraction-levels: temporal logics, first order logics, state machines, process algebras, synchronous languages, etc (see [11] for a survey). For a dual formalization (property + model) as ours, a model, a requirements language and several analysis methods are needed.

In order to include the timed component of a system, a model is needed which, in addition to reactive behavior, expresses the usual behaviors in real-time systems, like propagation delays, timers, deadlines, response times, etc. These real-time constraints have been incorporated in untimed state models by means of two major mechanisms: associating lower and upper bounds with transitions, or defining a finite number of clocks which proceed at the same rate and measure the elapsed time since they were reset (*timed automata* [12]). The later mechanism has been revealed as more flexible in real-time modeling.

Meanwhile timed automata (and its different versions) can be considered the standard formal state model, in the property approach, the situation seems not to be so clear: a variety of logics have been applied to the requirements specification [13]. Given that only branching real-time logics can be automatically verified (without relaxing punctuality [14]), we define a branching real-time logic which, moreover, is causal (closer to the user) and many-valued (supporting an incremental design).

As far as analysis is concerned, formal techniques assuming dense time have their early origin in [15]. In this work, *Alur et al* reach to a model checking algorithm for timed automata with real-valued clocks and the logic TCTL. The main idea behind this algorithm is the construction of an abstract finite state-space, called region graph, from the dense state space of a timed automata. Since obtaining a accurate finite model of the system was revealed as possible, many works have extended this solution to other kinds of analysis (strategy computation, schedulability analysis, synthesis).

Unfortunately, the number of states in such abstract space is exponential with elements in the model and the specification. Hence, algorithms based on explicit construction of the region graph are unlikely to perform efficiently in practice.

In order to overcome the state-explosion problem, different techniques have been applied: discrete representatives, symbolic computation, and minimization by time-abstractions. To sum up, discretization is not always correct; and symbolic analysis by means of characteristic formulae is not close to the user. For analysis methods as automatic as possible, minimization seems to us a promising solution. In [16], it is showed a catalog of the main time abstractions (simulations and bisimulations) and the kind of properties they preserve.

3 Formal basis

In this section it is outlined the formal basis we incorporate in the model of software process. We take a dual approach to design which has been massively adopted both in untimed and timed formal methodologies (for instance, KRONOS). Dual specification joins advantages of two broadly used styles: model- and property-oriented.

Property oriented: We define a real-time temporal logic for the requirements specification (SCTL-T, section 3.2). An iterative and incremental approach entails user information which is both incomplete and inconsistent, since the user gains knowledge about the system through the software process. For this aim, many-valued logics are an attractive solution.

Model oriented: We use a state-transition formalism for the design model of the system, since its operational style turns out to be more suitable than process algebras in the early phases of the life cycle when the structure sense is slight or it does not exist at all. We define a timed model (MUS-T, section 3.1), based on timed automata, which records design information generated all over the process. However, given that refinements are usually carried out over constructive specifications, we propose a semi-automatic translation from MUS-T into E-LOTOS [17].

Since part of the formal process means deciding what functionalities to be implemented in *software* and which in *hardware*, a state-transition model allows us to easily translate into hardware and, by means of E-LOTOS into software.

3.1 MUS-T

For modeling real-time systems we define MUS-T graphs (**T**imed **M**odel of **U**nspecified **S**tates), which are based on timed automata theory. Whereas atomic propositions in a control state of a timed automata are assertions being true or false, in an incremental approach, things will be true or false at the end (in final development), but in a transient process cycle things can be true, can be false, or can be non specified.

For this, we propose a model with dense branching-time semantics similar to the timed graph in [15] , but event-driven. In order to support incompleteness and consistency-checking, timed events (event + time guard) in a state of the model can be characterized as possible, forbidden or non-specified (unspecified). Transitions in the model are linked only to possible and unspecified events in a control state.

That is, for each control state and for each event identified in the system, time domain can be split in three specification-stage zones: possible zone, forbidden zone and unspecified zone. All legal runs of the system conform to the following rules:

- ➤ If current time valuation belongs to the forbidden zone, the event can not be taken in this timed state.
- ➤ If current time valuation belongs to the possible zone, the event can be taken in this timed state.
- ➤ If current time valuation belongs to the unspecified zone, this event can be characterized as possible or forbidden later in the software process.

In contrast to untimed models, timed ones define two types of possible evolutions from a state: discrete transitions and time transitions. Above characterization of events in a state of the model allows us to specify discrete progress in the system incrementally. However, in order to cover time evolution as well, invariants (predicates characterizing states where time can continuously progress) have to be formalized in a similar way. For this, each control state in the system is linked to a pair of invariants: evolution invariant and stop invariant. The evolution invariant determine the set of valuations that allow the state to advance time, similarly, the stop invariant determines the set of valuations making the system to take a discrete transition, or, if it does not exist, timelock. Time valuations which are included in neither of the above belong to the unspecified invariant, and they can evolve to the stop or the evolution invariant.

3.2 SCTL-T

The major motivation behind introducing a causal logic into requirements specification is to fill the gap between the natural language, in which users express themselves, and a formal specification of requirements. Causation respect three fundamental principles of requirements engineering: requirements specification is clear to the customers, clear to the developers and, by means of formal causation, it can be formally analyzed.

With this principle we define a causal temporal logic called SCTL-T (**T**imed **S**imple **C**ausal **T**emporal **L**ogic). It is a branching real-time logic with dense time semantics, that fits within the explicit clock real-time logics. To express the timed constraints we use specification clocks fixed by *freeze* quantification, and time predicates over these clocks.

Requirements in SCTL-T follow this pattern:

$$\textbf{Premise} \Rightarrow \otimes \textbf{Consequence}$$

This generic causal requirement establishes a causing condition (premise); a temporal operator which determines the applicability of the cause ($\Rightarrow \otimes$); and a condition which is the effect (consequence).

Apart from causation, SCTL-T is many-valued. The starting point of many-valued logics is that not "everything is true or false" (principle of bivalence). In general, many-valued logics are suitable to deal with both incomplete and inconsistent information obtained by the requirements capture, unfortunately, research in this area is deficient in theoretical results and, especially, in practical tools. Logic SCTL-T values outside bivalence principle are originated from unspecification in the model, and differences between implication and causality. Taking this into account, we have defined the six-value satisfaction relation of a SCTL-T formula over a timed state of a MUS-T graph.

Finally, although a causal logic captures the human sense of expressing requirements, the formal specification of a requirement in this logic may be difficult for users without mathematical foundations. We offer a graphical counterpart to the SCTL-T syntax which allows users to construct SCTL-T graphs from conditions over a state of the system, causal temporal operators relating them (characterizing a condition as a premise or as a consequence) and timing constraints demanded between whatever two state conditions.

4 The whole Software Process: introducing formality

Nowadays iterative and incremental development is a common practice in software engineering. In spite of its proven reduction in time to market, integration with formal methods area is still immature. Research has not furnished the formal basis and methodologies enabling this paradigm. Toward this field the main interest of my research group is directed, and my particular interest related to real time systems as well.

In this section we outline the whole proposed model for the software process. This model defines a software process which is both iterative and incremental; moreover,

it relies on formal basis (SCTL-T and MUS-T). Unfortunately, formal basis is not enough. A system is correct when it runs as it is desired, but formal methods can not demonstrate correctness in this sense, since they are based on formal models which could be unsuitable or incomplete; and, moreover, mistakes in the difference between user expectations and established requirements can appear. Although formal methods allow constructing precise specifications of the target system and requirements, it is necessary to use supplementary informal techniques like prototyping, simulation and testing.

On the one hand, an incremental and iterative life cycle, maybe prototype-oriented (explorative), enforces additional mechanisms: formal synchronization of software artifacts (models and requirements) between subsequent cycles (iterative); support for incomplete models (incremental); checking inconsistencies (requirements); etc. On the other hand, ensuring technology transfer enforces user-oriented helper mechanisms (like formal simulation, suggestion generation, mechanisms for easy requirements specification, etc.); and non-static analysis like prototype testing.

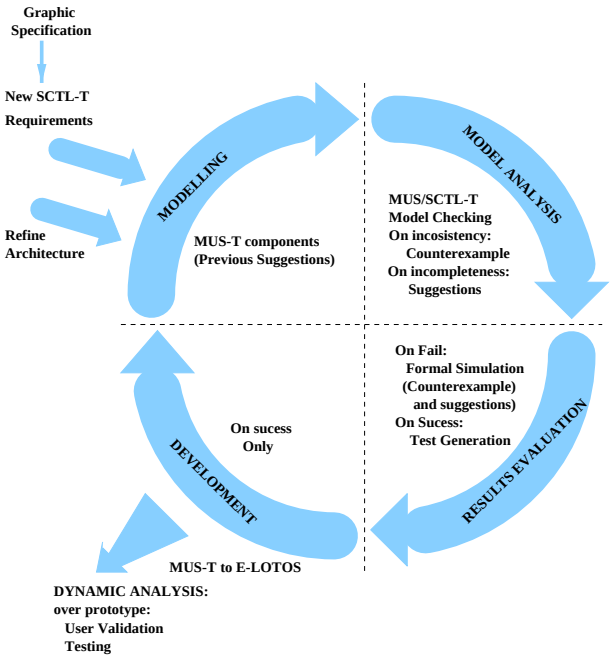

Figure 1. Model of software process

The model of software process we propose is showed in figure 1. In every iteration, the user identifies and specifies a set of requirements which lead to a growth in the system functionality. These requirements are verified in the current model, that one in the current cycle, in order to check: if the system already satisfies the requirements; if it is not able to provide, in a future design state, these requirements from the current design (inconsistency); or, if the system does not satisfy the requirements, but it is able to do it (incompleteness). The phases within a cycle are:

1 **Modeling**: requirements capture by means of SCTL-T graphic construction which is automatically translated into SCTL-T syntax; and change in MUS-T model, possibly incorporating suggestions in the previous cycle.

2 **Model Analysis**: new SCTL-T requirements specified in the modeling phase are verified over MUS-T components of the system. Analysis phase includes formal verification, as usual in FDT's, and even schedulability analysis due to real-time nature. Formal verification is carried out via model checking of SCTL-T requirements over composed MUS-T model by computing an abstract graph. The result of this verification process is the level of satisfaction of a SCTL-T requirement in a MUS-T graph. This satisfaction level shows, in a qualitative way, the system capability to fulfill requirements in future cycles, which depends on the amount of unspecification in the system and in the requirement. If a SCTL-T requirement is not satisfied:

   – consistency failure: It can not be satisfied now or in the future. A counterexample (execution trace) and the list of previous requirements in conflict are supplied.
   – completeness failure: The requirement can be fulfilled in the future. Suggestions are computed over the abstract graph, moving some valuations from the unspecified zone to the possible or forbidden ones, in order to fulfill new requirements. This can be viewed as an incremental synthesis.

   With respect to schedulability analysis, it is possible to reduce it to a specific model checking algorithm which computes a sample scenario where all tasks are served. Once again, the tasks can be schedulable, non schedulable, or incomplete to decide schedulability.

3 **Results Evaluation**: This phase allows the user to evaluate results in the analysis phase.

   – On success: A test suite is generated from the formal design.
   – On fail (inconsistency): User can simulate the counterexample in order to decide which of the conflicting requirements are error-prone.
   – On fail (incompleteness): User can simulate the current model with supplied suggestions in order to decide which suggestion conforms to wishes.

4 **Development**: MUS-T/ELOTOS translation (maybe architecture decisions) and prototype construction. Over the prototype, the user can validate the implemented system by testing. We use E-LOTOS in order to take advantage of our experience in this process algebra.

5 Current state and research goals

In the current state of my doctoral work, we have defined the formal basis for the methodology: MUS-T and SCT-L. With reference to the automatic mechanisms defined in the software process, the concrete goals which we have already reached are:

- ➤ Formal verification: A model checking algorithm for MUS-T and SCTL-T by computation of the region graph.
- ➤ Overcoming state-explosion: A minimization algorithm which builds the minimal stable partition of a MUS-T graph using a time abstracting bisimulation which preserves SCTL-T formulae.
- ➤ Formal simulation algorithm: The simulation algorithm makes a reachability analysis on the composed system. The purpose of this analysis as validation mechanism is two fold: guaranteeing sanity of the system by detection of timelocks, deadlocks and unreachable states which are real or potential (due to unspecification); and allowing developers to validate system requirements, and to explore alternatives in order to reach new requirements (by formal simulation of the system + suggestions). For the implementation of the simulation algorithm we use a time-abstracting simulation which preserves reachability.

In order to conclude this work, our immediate goals are: graphic notation for SCTL-T, suggestions computation, test generation, and definition of MUS-T into E-LOTOS translation. In the following paragraphs we outline the preliminary ideas related to these goals.

Graphic notation: As it is known, temporal logic formulae are complex to express and interpret. This complexity grows in explicit clock temporal logics due to the incorporation of named clocks in the formulas, making the specification hardly intuitive. With the aim of avoiding unreadability we intend to define a graphic notation which represents SCTL-T requirements as directed graphs in a way similar to the one defined in [18], but incorporating causal connectives.

Incremental Synthesis (Suggestions): We conjecture that the more general problem of synthesizing a MUS-T model from a SCTL-T requirement is undecidable, since the satisfiability problem is undecidable for real-time logics as long as the logic can express punctuality properties. As regard incremental synthesis, we are considering two approaches: game theory and bounded synthesis. Given a SCTL-T requirement, in order to provide (if completeness failure) suggestions to the user we are using the works on real time games (see [19] for its origins). In the game approach, we search for timed strategies which preserve the possible and forbidden elements in a MUS-T model, and restricts the choices of the unspecified ones so that the model satisfies a given SCTL-T requirement. For this purpose, we intend to design an algorithm as "on the fly" as possible, like the one recently showed in [20]. Although the former seems to us the most promising solution, we are investigating the feasibility of applying a bounded model construction algorithm similar to the one in [21]. In the bounded synthesis approach, we intend to provide an algorithm which, given a SCTL-T requirement and a source MUS-T model, synthesizes (if possible) a satisfying target model within given bounds on the number of clocks and constants used.

Test generation: At the moment we prefer not to release any comments of this part of the work since we are still considering suitability of different approaches in the state of the art.

MUS-T into E-LOTOS translation: In this part of the work we intend to extend the MUS into LOTOS translation in [6] to the real time version defined in my doctoral work.

6 Evaluation and preliminary conclusions

The main goal of this doctoral proposal is to define the formal basis and techniques needed to support iterative and incremental development for real time systems. This development paradigm has been massively incorporated with low-level formality in real time tools by vendors like Rational (RT Rational Rose [22]) , Verilog (ObjectGeode [23]), and IAR (IAR Visual State [24]). All of them include formality in design by means of FSM's, however, formal analysis is more exceptional and formalization in requirements is absent at all. It is needed more research effort in order to reach the same technology transfer success in the last two issues.

The evaluation of the contribution of my PhD thesis has to be carried out in two phases:

- ➤ Soundness: Evaluation of the correctness of the methodology.
- ➤ Measurement of the contribution: It is mandatory to evaluate the methodology with the common case studies tackled by academic tools in this area. However, this static evaluation (without process perspective) does not provide the real measurement of the value of my work since the main contribution is the formalization of the whole process. It would be advisable to address evaluation by means of success stories reported by vendors. In this kind of evaluation, it is important to take into account that we are implementing the methodology as a benchmark, and therefore we can come up against scale problems when it comes to deal with these success stories.

To conclude, we believe that higher-level formalization of software engineering in practice should involve approaching formal methods to current trends in software industry. At this point, we believe incremental approach is a major milestone. However, there could be some criticism about my doctoral proposal concerned to object orientation (OO). As well as formalizing an iterative and incremental process intends to avoid industry reluctance about formal methods, future directions in order to achieve this goal should invest more effort in OO paradigm. In this field we merely cite the works that include OO in real time logic specifications carried out in TRIO+ definition [25].

References

[1] John A. Stankovic. Real Time and embedded systems. *ACM Computing Surveys*, 28(1), 1996.

[2] Constance Heitmeyer. On the Need for Practical Formal Methods. In *5th International Symposium on Formal Techniques in Real-Time and Real-Time Fault Tolerant Systems (FTRTFT 98)*, number 1468 in LN CS, pages 18–26, Lyngby, Denmark, september 1998.

[3] John A. Stankovic. Strategic Directions in Real-Time and Embedded Systems. *ACM Computing Surveys*, 28(4), 1996.

[4] Jose J. Pazos Arias. *Transformación y verificación con LOTOS*. PhD thesis, Departamento de Ingeniería de Sistemas Telemáticos - Universidad Politécnica de Madrid, 1995.

[5] Alberto Gil Solla. *Diseño y verificación de sistemas distribuidos mediante la aplicación combinada de métodos formales*. PhD thesis, Departamento de Tecnología de las Comunicaciones - University of Vigo, 1999.

[6] Jorge García Duque. *Especificación, verificación y mantenimeinto de requisitos funcionales con técnicas de descripción formal*. PhD thesis, Departamento de Tecnología de las Comunicaciones - University of Vigo, 2000.

[7] ISO. *Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Tecnhique based on the Temporal Ordering of Observational Behaviour*. ISO/IEC/8807, Geneva, 1989.

[8] Rajeev Alur. *Techniques for Automatic Verification of Real-Time Systems*. PhD thesis, Department of Computer Science, Stanford University, 1991.

[9] A. Gollu, A. Puri, and Varaiya P. Discretization of Timed Automata. In *Proc. 33rd CDC*, Orlando, 1994.

[10] E. Asarin, O. Maler, and A. Pnueli. On Discretization of Delays in Timed Automata and Digital Circuits. In *Concur'98*, number 1466 in LNCS. Springer, 1998.

[11] Jonathan S. Ostroff. Formal Methods for the Specification and Design of Real-Time Safety Critical Systems. *Journal of Systems and Software*, 18(1), 1992.

[12] Rajeev Alur and David Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–236, 1994.

[13] Rajeev Alur and Thomas A. Henzinger. Logics and Models of Real Time: A Survey. In *Real Time: Theory in Practice*, number 600 in Lecture Notes in Computer Science. Springer Verlag, 1992.

[14] Rajeev Alur and Thomas A. Henzinger. Real-time Logics: Complexity and Expressiveness. *Information and Computation*, 104(1), 1993.

[15] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in Dense Real-Time. *Information and Computation*, 104(1), 1993.

[16] Stavros Tripakis. *L'Analyse Formelle des Systèmes Temporisès en Pratique*. PhD thesis, Universite Joseph Fourier, Grenoble, 1998.

[17] ISO/IECJTC1/SC21/WG7. Revised working draft on enhancements to LOTOS (v4). Technical report, 1998.

[18] H. Ben-Abdallah and I. Lee. A Graphical Language for Specifying and Analyzing Real-Time Systems. *Real-time Engineering Systems*, 5(4), 1998.

[19] O. Maler, A. Pnueli, and J.Sifakis. On the Synthesis of Discrete Controllers for Timed Systems. In E.W. Mayr and C. Puech, editors, *STACS '95*, number 900 in LNCS, pages 229–242. Springer, 1995.

[20] S. Tripakis and K. Altisen. On-the-fly controller synthesis for discrete and timed systems. In *World Congress on Formal Methods, FM'99*, 1999.

[21] Francois Laroussinie and Kim G. Larsen. From Timed Automata to Logic – and Back. Technical Report RS-95-2, BRICS. Department of Computer Science, University of Aarhus, january 1995.

[22] Garth Gullekson. Designing for Concurrency and Distribution with Rational Rose RealTime. White paper, Rational Corporation, 2000.

[23] Philippe Leblanc. The ObjectGEODE Engineering Process. White paper, VERILOG, february 1998.

[24] IAR Systems. *IAR visualSTATE Concept Guide*, october 1999.

[25] A. Morzenti and P. San Pietro. Object-Oriented Logic Specifications of Time Critical Systems. *ACM TOSEM - Transactions on Software Engineering and Methodologies*, 3(1):56–98, january 1994.