*Integrated Design and Process Technology,* **IDPT-2002**
Printed in the United States of America, June, 2002


# A FORMALIZATION OF UML STATECHARTS FOR REAL-TIME SOFTWARE MODELING

**Vieri Del Bianco, Luigi Lavazza, Marco Mauri**
**CEFRIEL and Politecnico di Milano**
**Via Fucini, 2 - 20133 Milano (Italy)**
**e-mail: {delbianc|lavazza|mmauri@cefriel.it}**
**Phone: +39-02-239541 Fax: +39-02-23954254**

**ABSTRACT**

The work presented here is part of a project that aims at the definition of a methodology for developing real-time software systems based on UML. In fact, being relatively easy to learn and use, UML is very popular, unlike formal methods. However, formal models provide developers with several benefits: they can be used for activities –like property verification, simulation, test case generation, etc.– which are vital for the development of real-time software. Such activities are more difficult, more error-prone or less effective when carried out on UML models, because UML is not formally defined.

In this paper we explore the possibility to formalize the part of UML that is more important for the specification of real-time behavior, namely the statecharts.

Actually, previous experiences indicated that UML statecharts have to be extended in order to cope with real-time issues. Therefore we aim at a formalization of a suitable extension of UML statecharts. For this purpose we do not develop a new formalism, instead we exploit the Timed Statecharts formalism proposed by Kesten and Pnueli.

The contributions of the paper are: the evaluation of the difference between Timed Statecharts and UML (properly extended for modeling real-time issues), a mapping of extended UML statecharts to Timed Statecharts, the application of the proposed approach to a case study.

**Keywords**: Real-time software, formal methods, state machines, statecharts, UML.

## 1 INTRODUCTION

UML is a de facto standard for documenting the specification and design of object-oriented systems. The continuously growing popularity of this notation has led software developers to use UML to model application domains that were originally out of the language scope. These domains include business processes, Web-based applications, information systems, component-based systems, etc. In general the rich set of diagrams provided by UML, together with a flexible extension mechanism, allow developers to model all the relevant features of software systems.

The structure of UML is quite formal, since it is based on a meta-language (defined using a reduced set of UML) which is used to specify the diagrams that compose UML. The meta-model of UML can be effectively used to expand and enhance UML according to specific needs, when the usual extension mechanisms of UML (stereotypes, tagged values, and constraints) are not enough flexible to achieve the desired expressiveness.

Although the syntax of UML is formally defined, the semantics is quite loose. This was not originally perceived as a problem, since the main purpose of the language was to be easy to use and to adapt to various needs: in order to achieve such goals, it was decided that the semantics of diagrams would be given informally. Currently this view is changing, as UML is growingly employed in new application domains that call for a better formalization, even at the expense of flexibility. This is usually the case for real-time, safety-critical and embedded systems. In such cases it is much preferable that the modeling notation is formal, as ambiguities and inconsistencies could have catastrophic consequences. Another issue in this field is the abundance of formal methods and supporting tools: although they have been proven useful to support the development of complex software, they are not much used in industry. This is due to the variety of formal methods and to the difficulty to use the tools, which often require sophisticated logical or mathematical skills.

In order to solve this kind of problems, we adopted a dual approach to real-time software development (Lavazza, Quaroni, Venturelli, 2001). In a first phase models are written in UML: this is expected to be a relatively inexpensive activity, since UML does not require a big learning effort, and it is well supported by tools. In a second phase UML models are automatically translated into one or more formal notations, which provide support to activities such as the verification of properties, test case generation, etc. In this way, developers exploit the advantages of formal notations while skipping the complex and expensive formal modeling phase. The optimal situation is when the derived models can be used by fully automatic tools: the modeler gets the results without even being aware of the underlying formal methods.

Of course, in order to apply such approach the semantics of UML must be formally defined. In previous work (Lavazza, Quaroni, Venturelli, 2001) we defined

UML's semantics implicitly, in terms of the translation from UML to TRIO (a first order temporal logic (Ghezzi et al., May 1990)). Here we address the problem of explicitly defining the semantics of UML, or at least of those parts of UML that are most relevant for real-time software. In particular, we consider object diagrams, class diagrams and state diagrams as the fundamental elements to model real-time systems. The former two types of diagrams do not need to be modified, while the state diagrams, which specify the behavior of the system, need to be suitably extended (as discussed in Lavazza, Quaroni, Venturelli (2001)).

Our approach is mainly based on Timed Statecharts (TS), as formally defined by Kesten and Pnueli (1992), i.e., we extend UML's syntax and semantics, and provide a mapping to Timed Statecharts' concepts. It is possible to identify four key parts of our approach:

- A mapping between UML and TSs is defined. In this step we use an extension of the standard definition of UML (UML Specification, 2001) that takes into account the need to express real-time constraints (Lavazza, Quaroni, Venturelli, 2001) as well as the need to keep close to TSs.
- UML Metamodel is modified to accommodate the previously identified changes. We call the resulting language UML+.
- XMI DTD is modified to represent UML+ diagrams and models by means of XML documents. We call XMI+ this extension of XMI.
- Translations to formal notations: a translator transforms the XMI+ documents equivalent to UML+ models into documents written in some formal notation. The latter can then be used by tools implementing formal methods. We developed two translators to convert UML+ models into TRIO logic formulas and Kronos timed automata, respectively. So a UML+ model can be verified without forcing the user to understand the underneath formal methods.

In this paper we illustrate the first point of the list given above. Translation from UML+ to TRIO is described in Lavazza, Quaroni, Venturelli (2001). Translation from UML+ to Kronos is described in Lavazza, Del Bianco, Mauri (2001).

This paper is organized as follows: Section 2 briefly illustrates the extensions to standard UML that were defined for real-time modeling. Section 3 illustrates how our extended version of UML is given a precise semantics by means of Timed Statecharts. Section 4 presents the application of extended UML to a case study involving the representation of a real-time system: the advantages deriving from the availability of a well-defined UML are illustrated. Section 5 briefly accounts for related work, while Section 6 draws some conclusions.

## 2 UML STATE DIAGRAM EXTENSION

The work presented is part of a larger project (DESS) whose aim is to employ UML in the specification of real-time systems. For this purpose we need a specification language that is both enough expressive to represent the features of real-time systems (with special reference to time constraints) and precise enough to support activities like property verification, test case generation, etc. The extensions of UML that are considered necessary for modeling real-time systems are described in Lavazza, Quaroni, Venturelli (2001) and Lavazza, Del Bianco, Mauri (2001). Here we summarize such extensions, which concern the state diagrams of UML.

A first limit of UML concerns the management of concurrent events. UML forces the events to be treated one at a time, thus making impossible to specify the behavior of the system when several events occur concurrently (unless the system behaves exactly as though the events occurred sequentially, of course). Our extension allows the modeler to associate a set of events to a transition, indicating that the transition is triggered by the concurrent occurrence of the set of events (see Figure 1).

A second problem with UML is that it does not allow the guards associated with transitions to contain references to events. Thus it is impossible, for instance, to specify that a given transition is executed if event A occurs while event B is not occurring. We simply removed this limitation by allowing guards to make reference to events, as illustrated in Figure 2.

In the specification of a real-time system it is often the case that a transition occurs at a time that depends on the occurrence time of another transition. It is therefore necessary to make reference to a given transition's occurrence time. In UML names may be placed on transitions to designate the times at which they fire. However these names can only be used within constraint expressions. We removed this limitation by allowing the modeler to make reference to these labels also in guards (see for instance Figure 4).

The last limitation we identified is probably the most relevant for real-time modeling. In fact the specification of a real-time system has often to prescribe that a given transition *must* occur in a well-defined time interval, maybe without specifying *exactly* when the transition has to occur. In other words, the specifications of real-time systems are often non-deterministic with respect to the times when transitions occur. The *After()* construct provided by UML is not suitable for modeling this type of situations. We removed this limitation by allowing the modeler to associate a time interval to a transition, thus indicating that the transition occurs at a time (not known a-priori) belonging to the interval (see Figure 4).

## 3 MAPPING EXTENDED UML TO TIMED STATECHARTS

### 3.1 A brief introduction to Timed Statecharts

The extensions described above clearly indicate the necessity to redefine the state diagram notation of UML. The Timed Statecharts proposed by Kesten and Pnueli (1992) suit well our needs and can be easily integrated in

the UML metamodel. It's important to observe that these changes cannot be expressed through UML extension mechanisms, thus they cannot be implemented as a UML profile. We need to introduce some new elements in the UML meta-metamodel as well as to modify the meaning of existing elements. The modified meta-metamodel defines the new language that we call UML+.

Timed statecharts extend the traditional statecharts by specifying time limits for the execution of transitions. The semantics is defined with reference to a dense time domain. This implies that the system may deal with arbitrarily close time events. Transitions are classified in two types: immediate ones and timed, or waiting, transitions.

Immediate transitions do not depend on time: they are (they actually *must* be) executed when a triggering event occurs. When no immediate transition is enabled, the time can flow with the state of the system remaining unchanged. On the contrary, timed transitions are independent from events. They are associated with a time interval specified giving a minimum and a maximum waiting time: the transition cannot be executed before the minimum or after the maximum waiting time. If no event changes the current state, the timed transition *must* be executed before the maximum waiting time.

Timed Statecharts allow to express negated events among the conditions which can determine the execution of a transition. Kesten and Pnueli (1992) propose a so called "weak time" semantics, i.e., transitions requiring no enabling event and with an associated delay $\tau$ and timeout $\upsilon$ may be performed (nondeterministically) at any time between $\tau$ and $\upsilon$. The concept of "step" is associated with the execution of an immediate transition; a reaction to an event may occur several steps after its generation, but still in the same timestamp. This kind of semantics is based on the fact that every generated event "persists" until the time does not flow. The time may flow only if all the transitions which were enabled by that event have been executed. In this way, several transitions triggered by the same event *e* are executed before the time becomes grater than the time of the occurrence of *e*.

### 3.2 Extension of Timed Statecharts

TSs define mechanisms and constructs that are similar to the ones described in Lavazza, Quaroni, Venturelli (2001). We therefore decided to define the semantics of our extensions in terms of TSs.

UML+ diagrams in Figure 1 and in Figure 2 already have a well-defined meaning in terms of TSs, therefore no specific discussion is necessary. However, we considered useful to introduce two modifications to the syntax of TSs, in order to facilitate modeling the behavior of a system:

- In UML+ we allow the modeler to associate both a set of triggering events and a time interval with transitions, as shown in Figure 3. This also has a well-defined meaning in TSs, provided that transitions like the one from S1 to S2 are interpreted as discussed below.
- The possibility to refer to a specific transition (state) in order to use the execution (entrance) time is illustrated in Figure 4.

In the example illustrated in Figure 4 the transition from S3 to S1 is modeled exploiting the labels (or timing marks) associated with the transition from S1 to S2 (T1) and the transition from S2 to S3 (T2). Transition T3 must be executed at a time t such that $T \leq t \leq T+(T2-T1)$, where T indicates the instant when state S3 was entered. Labels associated with transitions let us model precisely the time constraints governing the execution of transitions. For instance, we can specify that transition T3 has to be executed exactly after a permanence in S3 of duration (T2-T1) by associating it with the interval [T2-T1; T2-T1]. Note that the transition labeled T1 (for instance) can be executed several times. Therefore, it is necessary to specify which of its occurrences is associated with time T1. We assume that every timing mark indicates the latest execution of the associated transition.

We have also introduced the possibility of associating both triggering events and time constraints with a transition, as shown in Figure 5. The behavior of the diagram reported in Figure 5 is equivalent to the one reported in Figure 6. State S1.1 is entered exactly l time units after S1 (more precisely, S1.0) is entered. If event e1 occurs in next u-l time units state S1.2 is entered and event e1_bis is generated: as a consequence state S1 is immediately left and S2 is entered (we did not use a transition from S1.1 to S2 because inter-level transitions are not allowed in TSs). If e1 does not occur in a period of u-l time units after S1.1. is entered, then S1.2 becomes the new state. In this case the object will never leave state S1, since only event e1_bis can trigger the transition to S2, but the only chance to generate e1_bis has been missed. S1.2 can be considered as a "terminal state".

"Mixed" transitions are useful to model typical real-time constraints in a compact and intuitive way.

### 3.3 Adapting Timed Statecharts to UML

In the previous section we described the features (both syntactic and semantic) of TSs that can be readily incorporated in UML+. In this section we discuss how to adapt the features of TSs that do not match very well UML (or UML+). Note that the introduction of Timed Statecharts in UML doesn't require any modification of other diagrams.

A first fundamental issue is with the semantics of events, or how the evolution of the system is determined by events. In fact in UML events are processed sequentially, each event causing the execution of a run-to-completion (RTC) step. The next external event is processed after the current RTC step has completed, and the state machine has reached a stable state. In TSs there is no RTC concept, there is no queue of events nor dispatcher. TSs evolve according to a "chain semantics": according to this semantics it is possible to perform sequentially several transitions in the same step. This is also consistent with the interpretation of UML state

machines given in Lavazza, Quaroni, Venturelli (2001), therefore we adopted the event management principles of Timed Statecharts.

Other differences between the two notations are due to the fact that UML has been provided with several constructs –like transitions between states belonging to different levels in a hierarchy of states– which are not strictly necessary, but make it easier to model object-oriented systems. It is therefore interesting to define the meaning of these constructs by means of TSs.

Differences between UML state diagrams and TSs are summarized in Table 1. The most important differences mentioned in Table 1 are discussed in next subsection.

#### 3.3.1 Inter-level transitions

Inter-level transitions involve states belonging to different hierarchy levels, or to different composite states. For instance, transitions T2 and T3 in Figure 7 are inter-level transitions.

Figure 8 shows a legal TS (not employing inter-level transitions) whose behavior is equivalent to the state diagram reported in Figure 7. The conversion is straightforward and is therefore not discussed.

#### 3.3.2 Fork transitions

A fork transition has one origin state and multiple destination states, belonging to different concurrent regions of a composite state (see for instance Figure 9).

In the Timed Statecharts, a transition to a composite state implies that the initial sub-states of the concurrent regions are activated. Thus, transitions like the one depicted in Figure 9 can be described using TSs as illustrated in Figure 10.

#### 3.3.3 Final states

In UML a final state indicates that the evolution in a region of a composite state has reached a conclusion. When all the regions of a composite state reach a final state, a completion event is generated and the composite state is abandoned. TSs do not have final states, but can simulate them quite easily: final states are replaced by normal states; transitions to such states generate completion events. If the final states belong to concurrent regions, only the last region reaching a final state has to generate the completion event: this effect can be easily obtained using a counter.

#### 3.3.4 Entry and exit actions

Entry (and exit) actions are sequences of atomic actions that are executed every time a state is entered (exited).

In TSs, these actions can simply be associated with every entering (exiting) transition. Attention must be paid to preserve the correct activation sequence: entry actions are performed after the actions associated with the entering transitions; exit actions are performed before the actions associated with the exiting transitions.

## 4 A CASE STUDY: THE CSMA/CD PROTOCOL

In this section we apply UML+ to a case study described in Yovine (Oct. 1997). The presented use case is a typical real-time system, i.e., a system where timely response is extremely important, while data processing is marginal.

### 4.1 The CSMA/CD Protocol

In any broadcast network with a single channel the key issue is how to assign the use of the channel when many stations compete for it. A well-known protocol that solves this problem is the Carrier Sense, Multiple Access with Collision Detection, or CSMA/CD, protocol.

The CSMA/CD protocol works as follows. When a station has data to send, it first listens to the channel. If it is idle (i.e., no other station is transmitting), the station begins sending its message. However, if it detects a busy channel, it waits for a random amount of time and then repeats the operation. When a collision occurs, because several stations transmit simultaneously, then all of them detect it, abort their transmissions immediately and wait a random time to start all over again. If two messages collide then they are both lost.

The propagation delay of the channel plays an important role in the performance of the CSMA/CD protocol. It is possible that just after a station begins sending, another one becomes ready to send. If it senses the channel before the signal of the former arrives, it will find the channel idle and will start sending too. Hence, a collision will happen. Let s be the time for a signal to propagate between the two farthest stations. Suppose that at time t0 a station S0 begins sending a message. Thus, within the time interval [ t0 ; t0+s ), it is still possible that some station Si transmits, causing a collision. However, after time t0+s, the channel will be sensed busy by all the stations until the current message is delivered. Hence, the maximum time the channel could be sensed idle by any station after the beginning of a transmission is s.

Based on the fact above, we might think that a station that does not hear a collision for a time equal to s, could be sure that no other station would interfere. However, this conclusion is wrong. Due to the propagation delay, the noise burst caused by the collision could take a time s to arrive. In fact, in the worst case it would take 2s for a station to detect a collision.

In case of collision, each station waits randomly a time between 0 and 2s before trying again.

At last we consider the transmission time. Assume that only messages of equal length are sent and let l be the time spent to send a message. Then if no collision occurs, a message will be completely delivered in a time equal to l+s .

### 4.2 The model of the system

The static structure of the system is given in Figure 11. For simplicity we suppose the channel error-free and do not consider any buffering. The meaning of the attributes and methods reported in Figure 11 does not need comments.

The behavior of the system is described by the UML+ diagrams reported in Figure 12 and Figure 13. Such behavior is consistent with the textual description given in section 4.1.

### 4.3 Using the models

The model described above was created by means of a tool (derived from ARGO/UML, http:\\www.argouml.tigris.org) that is being developed at CEFRIEL. The tool exports the model in a format which is a suitable extension of XMI (see Lavazza, Del Bianco, Mauri (2001) for details concerning the definition of the extended XMI format).

The model was then translated by means of an automated translator (see Lavazza, Del Bianco, Mauri, 2001) into a set of Kronos automata. These where analyzed by means of the Kronos tool (Yovine, Oct 1997) in order to check the desired properties of the protocol. In particular, we have verified the following real-time properties, which were expressed using TCTL logic (Alur et al., 1993):

1. The system is NonZeno, that is, every finite run starting at the initial state is the prefix of some infinite divergent run, or equivalently, time may diverge in every state.
2. When a collision occurs, because two distinct senders transmit simultaneously, they both detect it at most s time units later.
3. When one of the senders begins transmitting, there exists a behavior that leads to a successful transmission without collisions such that the message is delivered in a time l.

In conclusion, the case study showed that it is possible to model a real-time system (namely the CSMA/CD protocol) employing a notation that though very close to UML is formally defined, so that the model can be analyzed by a model-checker.

## 5 RELATED WORK

### 5.1 Extensions of UML for consistency

Several efforts have been done to employ UML in the development of real-time software.

Douglass (1998) has shown how several features of UML can be exploited in the specification and design of real-time software. However, he has not solved the problems connected with the lack of formal semantics of UML. Moreover, he has not tackled difficult problems which –like the Generalized Railroad Crossing (see Lavazza, Quaroni, Venturelli, 2001)– cannot be modeled by means of plain UML.

Selic et al. (1999) have proposed the Real-Time Object-Oriented Modeling (ROOM) language, which has been merged with standard UML to form a proposal of 'UML for real-time'. This language is being used in several tools, like Rose for Real-Time and Anylogic. ROOM favours the design and coding activities, at the expenses of the specification phase. The language provides rich information on the structure of the system thanks to the facilities provided to describe active objects and their communications, and tools are available to translate the models into executable code which behaves like the models. The limits of ROOM are in the lack of constructs to describe non-trivial time constraints, as well as in the lack of formal semantics (semantics is hardwired in the ROOM virtual machine). As a consequence, a ROOM model cannot be analyzed in order to prove whether the model has desirable properties such as safety.

### 5.2 Formalization of UML

Other efforts are being spent in order to improve the formality of UML.

cTLA (Graw et al., 2000) adds temporal logic to existing UML diagrams. cTLA+ (Herrmann, 1997) is a compositional specification and verification technique based on Leslie Lamport's Temporal Logic of Actions TLA (Lamport, 1992). cTLA+ supports modular process type definitions and the composition of processes into systems. Processes can model components of an implementation. Moreover, they can represent modular logical constraints.

There are many other approaches aiming at a formalization of UML, all of these make UML more usable in critical, embedded and real time systems.

Two main efforts that are now converging in UML 2.0 (still in draft version) are ROOM (see above) and Catalysis (Francis D'Souza and Cameron Wills, 1998).

The approach in Catalysis is to use OCL (see UML Specification, 2001) constraints to eliminate most of the ambiguities that UML leaves unaccounted, i.e., OCL is used to formally define the diagrams in UML. Catalysis also introduces a notation to model components in a similar yet more accurate way than ROOM.

UML 2.0 adopts most of the notations of ROOM and some of Catalysis ones (in particular the usage of OCL to define diagrams).

Catalysis also inspired the Precise UML (pUML) project, which eventually evolved in the 2U (Unambigous UML) consortium. The approach in pUML is to formally express the UML Metamodel through OCL, with special attention dedicated to conformance and compatibility between different UML views, as well as to refinement.

The pUML project has developed a Meta-Modeling Framework (MMF) (Clark et al., 2001 and Álvarez et al., 2001) as a replacement of the original UML Meta Model, in order to eliminate many of the current ambiguities and deficiencies, and to make UML more modular and extensible. OMG is considering to fully adopt this approach in UML 2.0 (ad/00-09-01 and ad/00-09-02, UML 2.0).

Other approaches propose the usage of UML combined with a formal language. For example, in Bruel and France, Fusion approach and Object-Z language are integrated in UML. In Raghavan and Larrondo-Petrie a package to address real-time constraints is added to UML, but ambiguities in UML are not addressed.

Our work has similarities with both the efforts devoted to extending UML and those oriented to the formalization of the language. However our work is different from the mentioned approaches as UML is not integrated with other languages (as in cTLA or in the Object-Z based approach). Moreover, the formalization of UML+ based on Timed Statecharts allows the UML+ models to be translated in a relatively straightforward way into other notations (like TRIO logic or Kronos automata). Finally, we emphasize usability, and maintain UML as the unique "interface" to the model: neither the underlying formal model (based on TSs) nor the target notations (TRIO or timed automata) are visible by the modeler.

## 6 CONCLUSION

Our goal is to correct UML shortcomings (lack of real-time modeling facilities, ambiguity) while preserving the advantages (ease to use and learn). We defined a set of extensions oriented to real-time system modeling (Lavazza, Quaroni, Venturelli, 2001) and gave formal semantics to these extensions in terms of Timed Statecharts.

The result is a language which, although still easy to learn and use, is formally defined. Therefore we can translate real-time models into other formal notations in order to use existing tools. For instance, we were able to translate the extended UML model presented in Section 4.2 into timed automata and to analyze the latter with Kronos, in order to prove some properties of the modeled protocol.

In particular, when the formal methods can be applied automatically (as for Kronos) the user is not even aware of being using anything else than the extended UML language and the associated graphical editor. This is –in our opinion– a very important step for the diffusion of formal methods.

## ACKNOWLEDGMENTS

The work described here was carried out as part of the ITEA project DESS (www.dess-itea.org) (Software Development Process for Real-Time Embedded Software Systems). Project DESS is partly supported by MIUR.

## REFERENCES

OMG, XML Metadata Interchange (XMI) Specification, November 2000, Version 1.1, ftp://ftp.omg.org/pub/docs/formal/00-11-02.pdf.

OMG, Unified Modeling Language (UML) Specification, September 2001, Version 1.4.

Harel, D., 1987, Statecharts: A visual formalism for complex systems, Science of Computer Programming, 8:231-274.

Kesten, Y., and Pnueli, A., 1992, Timed and Hybrid Statecharts and their Textual Representation, Weizmann Institute of Science, In Formal Techniques in Real-Time and Fault-Tolerant Systems 2nd International Symposium.

Yovine, S., October 1997, Kronos, A Verification Tool for Real-Time Systems (Kronos User's Manual Release 2.2), Springer International Journal of Software Tools for Technology Transfer, Vol. 1.

Graw, G., Herrmann, P., and Krumm, H., 2000, Verification of UML-based real-time system designs by means of cTLA, In Object-Oriented Real-Time Distributed Computing (ISORC 2000) Proceedings, Third IEEE International Symposium on , 2000 Page(s): 86 –95.

Lavazza, L., Quaroni, G., and Venturelli, M., 2001, Combining UML and formal notations for modelling real-time systems, Joint 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE), Wien, September 10-14.

Lavazza, L., Del Bianco, V., and Mauri, M., December 2001, An introduction to the DESS approach to the specification of real-time software, CEFRIEL Technical Report RT 01125.

Ghezzi, C., Mandrioli, D., and Morzenti, A., May 1990, TRIO, a logic language for executable specifications of real-time systems. The Journal of Systems and Software, 12, 2.

Selic, B., Gullekson, G., and Ward, P.T., 1999, Real-Time Object-Oriented Modeling, Wiley.

Douglass, B.P., 1998, Real-Time UML, Addison Wesley.

Alur, R., Courcoubetis, C., and Dill, D.L., 1993, Model checking in dense real-time, Information and Computation, 104(1):2(34).

Bruel, J.M., and France, R., Transforming UML Models to Formal Specifications.

Raghavan, G., and Larrondo-Petrie, M., A formal UML Package for Specifying Real-Time System Constraints.

Francis D'Souza, D., and Cameron Wills, A., 1998, Objects, Components, and Frameworks With Uml: The Catalysis Approach, Addison-Wesley.

The precise UML group, http://www.cs.york.ac.uk/puml.

2U Consortium Unambiguous UML, http://www.2uworks.org/documents.html.

Clark, T., Evans, A., and Kent, S., 2001, Engineering modelling languages: A precise meta-modelling approach, http://www.2uworks.org/documents.html.

Álvarez, J., Evans, A., and Sammut, P., 2001, MML and the Metamodel Architecture, http://www.2uworks.org/documents.html.

Initial submission to ad/00-09-02 (UML 2.0 Superstructure), 22/10/2001, http://www.2uworks.org/documents.html.

Initial submission to ad/00-09-01 (UML 2.0 Infrastructure) ad/00-09-03 (UML 2.0 OCL), 22/10/2001, http://www.2uworks.org/documents.html.

Lamport, L., 1992, Hybrid Systems in TLA+, http://citeseer.nj.nec.com/lamport93hybrid.html.

Herrmann, P., Specification of Hybrid Systems in cTLA+, 1997, http://citeseer.nj.nec.com/herrmann97specification.html.

DESS project official Web site, http://www.dess-itea.org

**FIGURES**

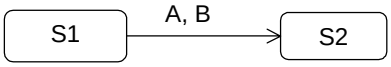


**Figure 1: A transition with multiple concurrent events.**

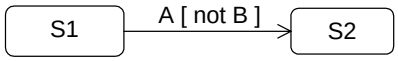


**Figure 2: A transition guard involving a negated event.**

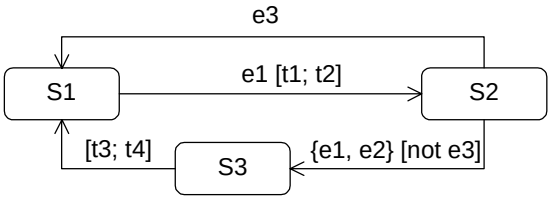


**Figure 3 A UML+ statechart.**

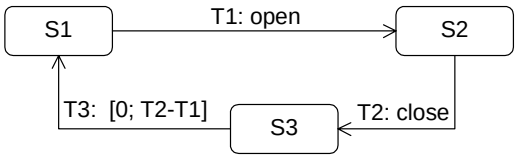


**Figure 4 A statechart with timing marks.**

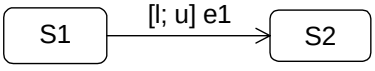


**Figure 5 "Mixed" transition, with both triggering events and time constraints.**

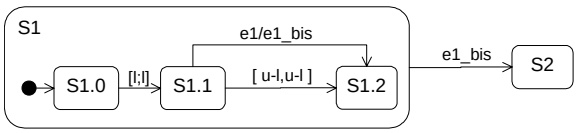


**Figure 6 A diagram with no mixed transitions, equivalent to the diagram in Figure 6**

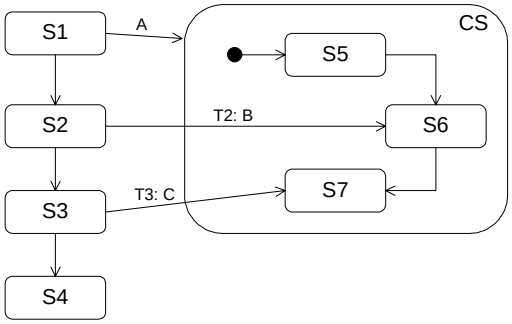


**Figure 7: UML diagram containing inter-level transitions.**

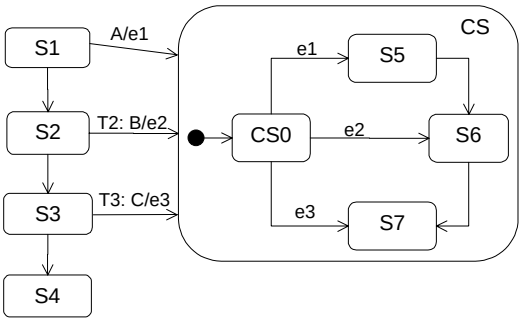


**Figure 8: Timed Statechart equivalent to the state diagram given in Figure 7**

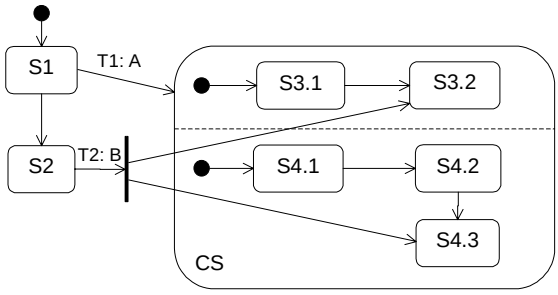


**Figure 9: UML diagram containing a fork transition**

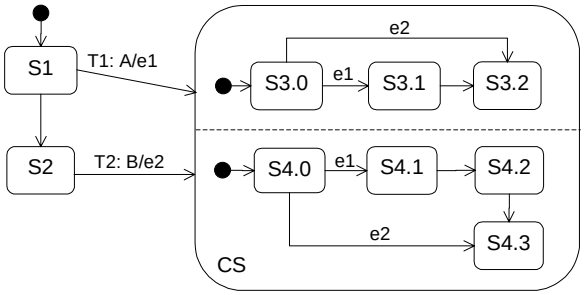


**Figure 10: Timed Statechart equivalent to the state diagram given in Figure 9**

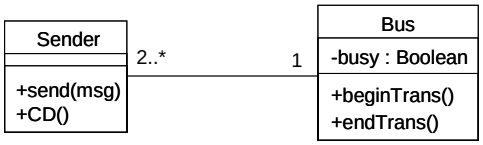


**Figure 11: Class diagram of the system**

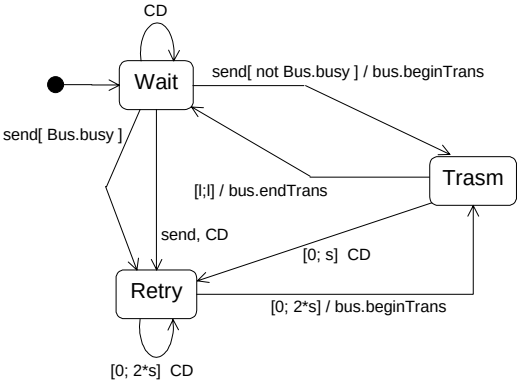


**Figure 12: UML+ statechart of class Sender**

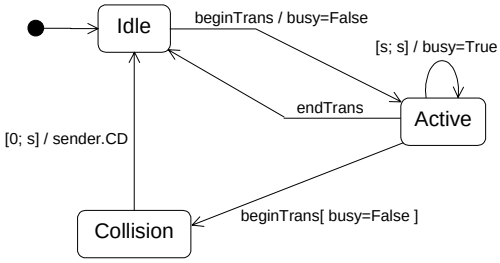


**Figure 13: UML+ Statechart of class Bus**

## TABLES

**Table 1: Differences between UML state machines and Timed Statecharts.**

| UML state diagrams | Timed Statecharts |
|---|---|
| Origin and destination states of a transition can belong to different hierarchy levels. | No inter-level transitions allowed. |
| Fork and Join transitions can be used to model concurrent processes and their synchronization. | No Fork and Join transitions. |
| Branch transitions have one origin and several destination states. | No branches: one transition must be defined for each possible condition. |
| A transition originating from the composite state S has lower priority with respect to those originating by substates of S. | Preemption can be used by a state with respect to any of its substates. |
| Deferred events can be executed after the current RTC step. | Deferred events can be modeled by means of timed transitions. |
| "History" pseudostates allow to go back to the most recent substate of a composite state. | As long as there are no inter-level transitions, history states are not needed. |
| Activities (having non-null duration) can be specified in states. They can be terminated by a transition. | There is no "activity" concept. |
| Completion transitions occur when the activity carried out in the considered state completes. | |
| Final states indicate that an object has completed its evolution, reaching a definitive state. | |
| Atomic actions (having negligible duration) can be associated to entering or exiting states. | Actions must be associated to transitions entering or exiting states. |