

# An Evaluation of Timed Scenario Notations

Jameleddine Hassine<sup>a</sup>, Juergen Rilling<sup>b</sup>, Rachida Dssouli<sup>c</sup>

<sup>a</sup>*Cisco Systems, 2000 Innovation Drive, Kanata, Ontario, K2K3E8, Canada*

<sup>b</sup>*Department of Computer Science, Concordia University, Montreal, QC, H3G1M8, Canada*

<sup>c</sup>*Concordia Institute for Information Systems Engineering, Concordia University, Montreal, QC, H3G1M8, Canada*

---

## Abstract

There is a general consensus on the importance of good Requirements Engineering (RE) for achieving high quality software. The modeling and analysis of requirements have been the main challenges during the development of complex systems. Although semi-formal, scenario driven approaches have raised the awareness and use of requirement engineering techniques, mostly because of their intuitive representation. Scenarios are a well established approach to describe functional requirements, uncovering hidden requirements and trade-offs, as well as validating and verifying requirements.

The ability to perform quantitative analysis at the requirements level supports the detection of design errors during the early stages of a software development life cycle. Thus, it would reduce the cost of later redesign activities. In order to achieve this goal, non-functional aspects and in particular time-related aspects have to be incorporated at the software requirement phase. This is essential in order to correctly model and analyze time dependent applications at early stages in system development.

The widespread interest in time modeling and analysis techniques provides the major motivation for our paper. The objective of the article is to provide readers with sufficient knowledge about existing timed scenario approaches to guide them in making informed decisions to when and how time aspects can be incorporated in their development process. In order to support this process, we present a comprehensive classification, evaluation and comparison of time-based scenario notations. In order to evaluate these existing notations, we introduce a set of eleven time-related criteria and apply them to categorize and compare forty seven scenario construction approaches.

*Key words:* Requirements engineering, functional, non-functional, model, analyze, timed scenarios, classification.

---

## 1. Introduction

In the early stages of development processes, system functionalities are defined in terms of informal requirements and visual descriptions. Scenarios are known to help requirements engineers not only to elicit such functional requirements, uncovering hidden requirements and

---

*Email addresses:* [jhassine@cisco.com](mailto:jhassine@cisco.com) (Jameleddine Hassine), [rilling@cs.concordia.ca](mailto:rilling@cs.concordia.ca) (Juergen Rilling), [dssouli@cse.concordia.ca](mailto:dssouli@cse.concordia.ca) (Rachida Dssouli)

trade-offs, but also to allow for the comprehension and validation of requirements. Scenario-based approaches go beyond the requirement phase by covering the whole software development life cycle and improving the communication among various stakeholders in establishing the requirements. The exact definition of a scenario may vary depending on purpose, contents and used semantics [RP96], but most definitions include the notion of a partial description of system usage as seen by its stakeholders [RR98]. With the advent of object-oriented design modeling more than a decade ago, the concept of *use cases* [JBR99] becomes a widespread practice for capturing functional requirements. Most authors agree that, in broad terms, use cases and scenarios are descriptions of a sequence of actions or events of some generic task which the system is meant to accomplish. However, there is no agreed distinction between the meanings of use case and scenario. Jacobson et al. [JEJ94], define use cases as follows: “a use case is a sequence of transactions in a system whose task is to yield a measurable value to an individual actor of the system.”. In UML context, Rumbaugh et al. [RJB99] define a scenario as a sequence of actions that illustrates behavior. A scenario may be used to illustrate an interaction or the execution of a use case instance [RJB99]. A slightly different distinction between “use cases” and “scenarios” is stated by Maiden et al. [Mai98]. The authors treat use cases as a collection of actions and the temporal rules that govern how the actions can be linked together. In contrast, a scenario is one sequence of events, the ordering of which is tied to the start and the end events or actions in the use case.

The design and implementation of distributed real-time systems is often dominated by non-functional considerations like timing, distribution and fault tolerance. As a result, it is increasingly recognized that non-functional requirements should be considered at the earliest stages of system development life cycle. Among these, time-related requirements have received considerable attention by the modeling community with several timed extensions of various notations and tools. For instance, the Unified Modeling Language (UML) [OMG07b], as a standard, considers real-time aspects in the profile for Schedulability, Performance and Time [OMG02].

The ability to model time constraints at the system requirement level not only facilitates the task of moving towards real-time design, but ultimately supports the early detection of errors through automated validation and verification of timing requirements. Thus, it would reduce the cost of later redesign activities in case some of the required time constraints are not met.

In this paper we present a state-of-the-art survey and evaluation of timed scenario-based approaches. The widespread interest in time modeling and analysis techniques, provides the major motivation for our paper. We, in particular, focus on the need for a comprehensive classification, evaluation and comparison of time based scenario notations applicable during requirement engineering. This paper serves different purposes:

- A set of eleven time-related categorization criteria is provided. These criteria are introduced in Section 3.
- A comprehensive survey of time-based scenario approaches and their notations is presented. Fifteen timed scenario notations and a total of forty seven construction approaches are reviewed in Section 4.
- We present a detailed comparison of selected time scenario-based construction approaches. Tables 1-8, which summarize this evaluation, can be used as an index when adding time aspects in the requirement phase. Section 5 provides an evaluation summary.

Given the scope of this study and the number of surveyed notations and techniques, this paper can neither offer illustrative tutorials or examples, nor provide an empirical comparison based on

a common case study. Our decision to provide multiple illustrative examples instead of a single representative example is dictated by the fact that existing notations support different time and untimed aspects. A common case study would have to be a very basic example, covering only the most fundamental aspect of a timed scenario notation. Therefore, we do believe that such a rather trivial common case study would not provide any meaningful additional insights. However, we believe that this paper will provide readers with sufficient knowledge about existing timed scenario approaches to help them make informed decisions about incorporating time aspects in their development process.

The remainder of this paper is organized as follows. The next section provides an overview of existing scenario classification approaches. In Section 3, we define a collection of eleven criteria that will help categorize and compare many timed scenario notations. Section 4 reviews fifteen timed scenario-based languages and forty seven construction approaches using the defined criteria. A summary of the classification is given in Section 5. Finally, conclusions are drawn in Section 6.

## 2. Classifications of Scenario Notations

Many classification approaches have been proposed to categorize and compare scenario notations. In what follows, we provide a brief overview of six comparison studies.

- Cockburn [Coc97] used four dimensions to use case descriptions, namely purpose, content, plurality, and structure. Purpose can be either for stories (explanations) or for requirements. Content can be contradicting, consistent prose, or formal content. Plurality is either one or multiple, in a way similar to multiplicity. Structure can be unstructured, semi-formal, or formal.
- In a European industrial survey, Arnold et al. [AEG<sup>+</sup>98] proposed a classification taxonomy for scenario usage in industrial projects. Their criteria are grouped under five main divisions: project properties, scenario contents and representation, goals, process, and experiences and expectations. They surveyed twelve industrial projects from various domains (telecommunications, sales, medical, software development, insurance, banking) where scenarios are used.
- Rolland et al. [CRH98] proposed a scenario framework based on four different views allowing to capture a particular relevant aspect of scenarios. Each specific scenario is characterized according to these four views.

The proposed four views are:

- **Content View:** It defines the kind of knowledge described in a scenario. Scenarios may, for instance, focus on the description of a system functionality or describe a broader view in which the functionality is embedded into a larger business process with various stakeholders and resources bound to it. Content view helps address the following concern: what part of the work activity is captured in a scenario ?
- **Form View:** It deals with the expression mode of a scenario. It specified whether scenarios are formally or informally described, come in a static, animated or interactive form.

- **Purpose View:** It is used to capture the role that a scenario is aiming to play in the requirement engineering process. Describing the functionality of a system, exploring design alternatives or explaining drawbacks or inefficiencies of a system are examples of roles that can be assigned to a scenario. The purpose view helps address the following concern: For what usage in the design process is it captured ?
- **Life-cycle view:** It suggests to consider scenarios as artifacts existing and evolving in time through the execution of operations during the requirements engineering process. Creation, modification refinement or deletion are examples of such operations. The question of persistency versus transience is also addressed in this view.

This framework uses a faceted classification method where a set of facets is associated to each view. Facets are considered as viewpoints or dimensions suitable to characterize and classify a scenario according to this view. For example, the description facet in the form view is a facet helping to classify scenarios according to the medium and the language used for their description. A facet has a metric attached to it. Each facet is measured by a set of relevant attributes. For instance, the description facet is measured with two attributes, namely the medium attribute and the notation attribute.

- Chance and Melhart [CM99] introduced a taxonomy of scenarios with the objective to improve understanding of scenarios and their usage. This taxonomy is organized into a hierarchy according to the purpose of the scenario. Scenarios can describe basic functionality (operational scenario), describe abnormal conditions (failure scenario), help evaluate system response (performance scenario), aid in requirements analysis and elicitation (refinement scenario), or be used to explain the system behavior to others (learning scenario). Each type of scenario is described in terms of key attributes: (1) Description: it describes how this category of scenarios differs from all other categories; (2) Creators/Users: it lists the architects and developers most likely to create or use the scenario category; (3) Information Needed: it lists the information that is useful to create scenarios of this category; (4) Uses: it lists the most common uses of scenarios of this category.
- Amyot et al. [AE03] defined nine criteria to categorize and compare fifteen scenario-based notations applicable to the telecommunications domain.

The proposed criteria are:

- **Component focus:** Scenarios can be described in terms of communication events between system components or independently from components, in a pure functional style.
- **Hiding:** Scenarios could describe system behavior with respect to their environment only (black-box), or they could include internal (hidden) information as well (gray-box).
- **Representation:** Scenarios can be described in various ways, for instance with semi-formal pictures, natural languages, structured texts, logic, grammars, trees, state machines, tables, visual paths, and sequence diagrams.
- **Ordering:** Scenarios represent a collection of events that can be ordered sequentially or causally.

- **Time:** Support for expressing time constraints with appropriate data types and evaluation mechanisms.
  - **Decomposition:** Decomposition can be hierarchical (which improves scalability) or be achieved through dependencies (e.g. references, contains, etc.).
  - **Abstraction:** An abstract scenario is generic, with formal parameters, whereas a concrete scenario focuses on one specific instance, with concrete data values.
  - **Identity:** Scenarios can focus on one actor (useful for component-oriented implementations) or target many actors at once (useful when describing end-to-end situations).
  - **Dynamicity:** A scenario notation is dynamic when it enables the description of behavior that modifies itself at run-time, otherwise it is said to be static.
- Liang et al. [LDD06] have presented a comparative survey of 21 approaches found in the literature based on two sets of comparison criteria. One set of criteria is for assessing approaches from a user's perspective. The other set of criteria compare the approaches from a more technical perspective, by focusing on the synthesis of scenario-based models into state-based models.
    - **Criteria Relevant From a User's Perspective:**
      - \* **Intended use.** Approaches are classified as intended for analysis only, or for both analysis and code generation.
      - \* **Source notation.** The choice of source notation (syntax and semantics) may influence the users' ability to describe scenarios with different levels of expressiveness as well as affect the synthesis algorithms.
      - \* **Support of composition mechanism.** By using composition mechanisms some scenario notations have the ability to express the behavior of complex systems more comprehensively.
      - \* **Support of parallelism.** Parallelism is either implicitly supported by means of the underlying semantics or explicitly supported by means of parallel composition constructs. By supporting parallelism, scenario notations can describe reactive systems more realistically.
      - \* **Target notation.** The choice of a target notation is mainly influenced by the intended use as well as the previous experience of the designer.
      - \* **Model type.** This criterion is closely related to the intended use. For instance, an approach may concentrate on deriving a set of object state models, or try to generate one single global state model for the whole system.
      - \* **Synthesis path:** Scenario-based models are categorized into either basic scenarios (BS) without using any composition mechanisms, or global scenarios (GS) obtained through the composition of BSs. These scenario-based models are synthesized into state-based models: object state machines (OSM) and global state machines (GSM) which are composed of OSMs. Four synthesis paths from scenario-based models to state-based models are proposed: BS→OSM, BS→GSM, GS→OSM, and GS→GSM).
      - \* **Degree of automation:** The generation of state-based models from scenario-based models can either be semi-automatic or fully-automatic.

- \* **Tool support:** Specifies whether a synthesis approach is supported by a tool.
- **Criteria Relevant From a Technical Perspective:**
  - \* **Inter-scenario relationships.** The authors have identified five different ways to identify the inter-scenario relationships. The designers can implicitly infer the relationships from the scenarios by using events or from the semantics of scenario notations. Additionally, the designers can explicitly define the relationships among scenarios by composition mechanisms, conditions, or a combination (hybrid) of both composition mechanisms and conditions.
  - \* **Consistency check.** Approaches may allow checking the consistency of scenario-based models before or during the synthesis processes.
  - \* **Completeness check.** Completeness checks on implied scenarios (extra behaviors) or missing scenarios (fewer behaviors) may be provided by the approaches.
  - \* **State space reduction.** Depending on how the inter-scenario relationships are identified, an approach may merge states and thus reduce the state space to various degrees.

The classification approaches listed above either overlooked time aspects or had a very limited coverage of time-related mechanisms.

### 3. Time-based Evaluation Criteria

Contrary to the approaches presented in the previous section, time-related aspects play a central role in our proposed classification approach. In this section, we propose a collection of eleven criteria that will help categorize and compare many timed scenario notations.

#### 3.1. Timed-action/event enabling

Intuitively, an action/event is enabled (i.e. offered) when the execution of its predecessor is completed. However, a time constraint can be specified to define when an action/event is offered relative to the execution of its predecessor action/event. Three types of enabling can be defined [BG06]:

- Simple enabling: The instant an action/event becomes enabled (e.g. available to the process and its environment) may be associated with a time constraint. For example, an action  $b$  can be taken at any time 5 time units after action  $a$ . However, no upper bound can be imposed on enabling.
- Initiation and termination of enabling: both lower and upper time bounds can be imposed on enabling. Thus, an action/event is continuously offered within a specific time interval. For example, an action  $a$  can be enabled immediately and will be offered for 5 time units.
- Punctual enabling: This type is a restriction of the *Simple enabling*. An action/event is enabled with respect to its associated time constraint then the enabling retracts if the action is not taken. For example, an action  $b$  is offered 5 time units after  $a$  and should be taken instantaneously, otherwise the enabling is retracted. Punctual enabling is used as abstractions of real-world systems. An important class of applications that use punctual enabling are those employing periodic behavior.

Another notational alternative to express enabling classes is:

- Delays: An action/event can be explicitly delayed by using a delay operator, such as  $\delta t$  or *WAIT* $t$ , used in timed process algebra [LL94]. Using such operators, lower bounds on enabling can be defined.

### 3.2. Instantaneous (atomic) vs. Durational Actions

Actions can be instantaneous (atomic) and the passage of time is explicitly modeled by a special *tick* action [RR88, NS94]. In an interleaving semantics, concurrency is reduced to *non-determinism* where the behavior of a system that performs two actions  $a$  and  $b$  concurrently is considered the same as the behavior of a system that either does  $a$  followed by  $b$ , or  $b$  followed by  $a$ .

Alternatively, actions can take a given amount of time, called *duration*, to be performed [GRS95, CFP01]. Hence, the time passes due to the execution of these actions. Approaches that use durational actions may support true concurrency. However, in the context of durational actions, Hoare [Hoa85] suggests that time-consuming actions should be represented by a pair of events, the first denoting its start and the second denoting its finish.

### 3.3. Absolute vs. Relative Time

The time of occurrence of an action/event of a system execution can be related to the starting time of the system execution, in this case, the time features as *absolute* [Cor00]. Alternatively, they can be relative to the execution of a causally preceding action, in this case, the time features as *relative*. In this case, the preceding action/event enables (directly or indirectly, i.e. via some intermediate events) the subsequent action/event.

### 3.4. System Clocks: Local vs. Global; Physical vs. Logical

The elapsing of time in a system can be modeled by a unique centralized global clock that increases uniformly. However, in a distributed system, many local clocks are used to track time in different locations. This raises the problem of clock synchronization [Mes90]. To address this issue, Lamport [Lam78] has introduced the concept of logical clocks, so partial ordering of events can be obtained without recourse to any physical “real” time.

### 3.5. Urgency

Urgency is a well-accepted and extensively documented time requirement [NS92, HR95, Sin04, BST98, BS00]. Urgency offers an abstraction that can be used to influence the behavior of a system as time progresses. It allows for expressing assumptions on the environment and on the underlying execution system, such as action durations, communication delays, or time constraints on external inputs. We distinguish two main approaches:

1. *Action Urgency*. This type of urgency is studied extensively in the process algebra theory [ISO97]. Three main approaches have been identified [BG06]:
  - *Urgent Actions*. In this approach, all observable and unobservable actions (i.e. internal to the system) are interpreted as urgent. Hence, it is possible that observable actions will become urgent, but will be prevented from executing by an environment that is not offering the action. This leads to a *timelock* situation in which time is not able to pass. Urgent actions are largely rejected in timed process calculi.

- *Explicit Urgency Operator*. In this approach, a specific operator, such as *urge* [BL92], is used to make an action urgent. *urge* is associated to an action and is placed in the beginning of a process behavior. This approach constraints the environment more selectively but timelocks can still occur [BG06].
- *Urgent Internal Actions*. This approach restricts urgency to internal actions (i.e. unobservable to the environment) and all observable actions are interpreted as non-urgent. Internal actions can always execute urgently without the possibility of *time-lock* since they are not controlled by the environment. This approach is now the most common approach in timed process algebra [ISO97]. Variants of this class of urgency are *Maximal progress*, *asap*(as soon as possible) and *Minimal Delay*.

2. *Transition Urgency*. A transition can be regarded as urgent, if it is taken or disabled before time progresses. There exists three main types of transition urgencies:

- *Eager transitions*. Eager transitions are urgent as soon as they are enabled, i.e. they never wait. They have to be executed as soon as possible and time should not progress as long as an eager transition is enabled.
- *Lazy transitions*. Lazy transitions do not prevent time progress in any system state, i.e. they can wait. Whenever a lazy transition is enabled, it can be taken, or likewise time can progress and possibly disable it.
- *Delayable transitions*. Delayable transitions are a combination of both eager and lazy transitions. They can wait, but they become urgent when time progress would disable them.

The distinction between these three types of transitions is depicted in Figure 1 [Sin04].

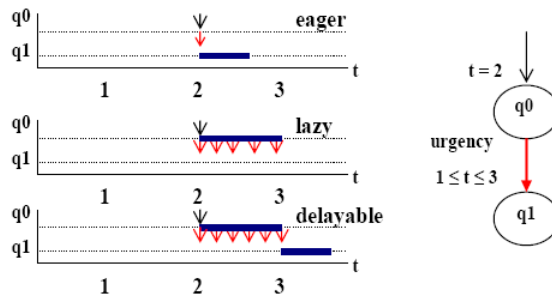


Figure 1: Transition Urgencies [Sin04]

### 3.6. Time Domain

The expression *discrete* or *continuous time* often refers to the empirical description of a so-called *physical time*. There are three types of physical time:

- *Discrete Time*. Empirical time is composed of indivisible instants, such that the passage from one instant to another implies an irreducible jump. In this sense, discrete time is a model isomorphic (i.e. structurally identical) to a discrete series of natural numbers.

This type of time model is appropriate for *synchronous systems*, where all the components are synchronized by a single common clock. This model has been successfully used for reasoning about the correctness of synchronous hardware design especially synchronous digital circuits, where signal changes are considered to change exactly when a clock signal arrives. One of the advantages of this model is that it can be transformed easily into an ordinary formal language. Each timed trace can be expanded into a trace where the times increase by exactly one at each step, by inserting a special *silent* event as many times as necessary between events in the original trace. Once this transformation has been performed, the time of each event is the same as its position [EMCGP99].

- *Continuous Time*. The jump between two instants is a smooth and uninterrupted process. In this sense, the model of continuous time is isomorphic to a continuous series of non negative real numbers. Continuous time model is appropriate for *asynchronous systems*, because the separation of events can be arbitrarily small. This ability is desirable for representing causally independent events in an asynchronous system. Moreover, no assumptions are needed about the speed of the environment when this model of time is assumed.
- *Dense (but countable) Time*. A model of dense time is isomorphic to a dense series of non negative rational numbers, meaning that there is always a rational number between any two rational numbers.

### 3.7. Time Representation/Measurement

Time can be represented by three classes: *point-based*, *interval-based* or both of them. In point-based models, the elementary units are points in a time space. Each event in the model has its associated time point (a single concrete time value). The time points can be arranged according to some relations such as *precede* or *after*. The ability to reference the time instant at which an action/event occurs is important in certain classes of real time specifications. In interval-based models, two different approaches can be considered. In the first, intervals are assumed to consist of points, and hence, the corresponding systems may be considered as models of point-based time theories [BK94]. The second approach takes intervals (i.e. ranges of time values within given bounds) as primitive objects, without any reference to the definitions of internal-point structures. Interval-based models are mainly based on the relations defined by Allen [All81]: *before*, *meets*, *overlaps*, *finishes*, *during*, *starts*, *equals*.

Time observations are described by measurements. Measurements are used to observe the delay between the enabling and occurrence of an event/action (for relative timing) and to measure the absolute time of the occurrence of an event/action (for absolute timing).

### 3.8. Time expressiveness (Timed Constructs/Constraints)

A timed scenario language is expected to offer a set of constructs that help:

- Express time dependent system behavior, such as execution time of tasks and actions. These are often modeled by means of timers or explicit access to a system clock.
- Express time constraints on the internal system execution such as end-to-end delays of the system.
- Express time related assumptions on the external environment of the system, mainly response time and inter occurrence time of stimulus.

### 3.9. Informal vs. Semi-formal vs. Formal Semantics

The nature of semantics (i.e. informal, semi-formal or formal semantics) offered by an approach, may impact the users' ability to precisely understand and reason about models expressed in the notation. Furthermore, it may also affect their ability to conduct time based quantitative analysis.

The choice of source notation (syntax and semantics) may influence the users' ability to describe scenarios with different levels of expressiveness as well as affect the synthesis algorithms.

### 3.10. Time Analysis and Verification

The quantitative analysis of requirement models allows the early detection of potential behavioral (time dependent) and performance problems. This criterion aims to identify what kind of timing analysis (such as validating timing assignment, verifying timing consistency, etc.) scenario notations offer. Different approaches have proposed algorithms and methods to analyze and ensure that timing requirements are met. These approaches can be classified as follows:

- Model-Based Scheduling Analysis: The first contribution to real-time scheduling theory was made by Liu and Layland [LL73], who developed optimal static and dynamic priority scheduling algorithms for hard real-time sets of independent tasks. Since then, much work on schedulability analysis has been done which includes various extensions of these results [JP86, MGH94, TBW94]. Schedulability analysis techniques are based, amongst others, on worst case execution times (WCET) [PB00] and stochastic task execution times [GL99].

Note: End-to-end system behavior description is necessary to conduct schedulability analysis.

- Formal Verification Approaches: These approaches are based on a translation of the timed requirement model into a formal description technique supporting time, such as timed automata [AD94]. The resulting models are checked against timing requirements using formal verification techniques, such as model checking [EMCGP99].

### 3.11. Specification Executability and Tool Support

Using scenario approaches to describe timing requirements may lead to:

- Executable specifications with appropriate operational semantics that can be simulated and tested.
- Off-line specifications that are not testable but offering a rich expressive power of time constraints.

The usefulness of a timed scenario notation may be directly related to the kinds of tools it provides to support design, analysis, and executable system generation.

Assessing the level of executability and tool support represents an important criterion in selecting the appropriate time scenario notation.

#### 4. Survey of Selected Timed Scenario Languages

In Section 2, we have presented a literature review of untimed scenarios classification approaches [LDD06, AE03, CRH98, Coc97, AEG<sup>+</sup>98, CM99]. In this section, we survey forty seven time-based construction approaches (corresponding to fifteen timed scenario notations) based on the eleven evaluation criteria specified in the previous section.

A summary of the evaluation is given in tables 1-8. For each construction approach, we indicate explicitly the selected criteria. For example, we specify whether a particular approach uses discrete or dense time as *time domain*, global or local as *Clocks*, relative or absolute as *time model*, etc. In addition, we have added the following terminology to cover different cases where an assessment using explicit values was not applicable:

- : Absence of the feature: The language does not support the feature.
- : Weak to basic existence: The language has a very basic support of the feature.
- + : Rich set of features: The language provides a fair/rich support of the feature.
- ? : Not specified: The information about the criteria is not available.
- N/A : Not applicable: The feature is not applicable to the notation.

##### 4.1. Timed (variants of) MSCs

Basically, timing constraints in (variants of) MSC notations are expressed using *timers* [IT96, AHP96], *delay intervals* [AHP96, MS93] and *timing markers* [BRJ96, LL99a].

- *Timers*. Timer support in the early standard version of MSC language (MSC-96 [IT96]) is very basic. A timer can be set to an optional duration, reset to zero, and observed for timeout. Figure 2 illustrates the stand-alone occurrences of the timer events in the MSC-96 standard as well as combined timer events. A timer set event (labeled by the timer name and with an identifier for the duration) is denoted by an hourglass symbol attached to the instance axis by means of a horizontal or bent line. A timer reset event is denoted by a cross which is attached to the instance axis by means of a horizontal or bent line. A timeout is represented by an hourglass symbol which is attached to the instance axis by means of an horizontal or bent arrow from the hourglass symbol to the instance axis.

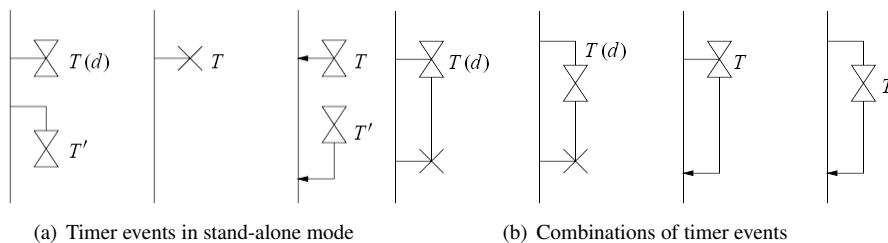


Figure 2: Timer Handling in MSC-96 [IT96]

The current standard (MSC-2004 [IT04]) describes some syntactic and semantics changes and refinements. The *set* event has been renamed to *start timer* and *reset* has been renamed

to *stop timer*. Timer duration is specified with an interval having an optional lower bound and an optional upper bound allowing the timer to expire within the specified interval. The upper bound for a timeout can be defined to be infinity which is represented by the keyword *inf*. A timer can be used to express a maximal delay between two or more consecutive events in one process. A timer cannot be shared among concurrent processes in a basic MSC (*bMSC*).

- *Delay Intervals*. Delay intervals are used to express three types of timing constraints: (1) *event-associated intervals* [MS93] which are denoted as an interval that is associated with an event (i.e. minimal and maximal delays within which the event should occur with respect to any previous event, whenever it occurs in an execution trace); (2) *message delivery delays* [AHP96, MS93] indicate the minimal and maximal delays allowed from the moment a message is sent until it is received (expressed as a time interval over a message arrow); and (3) *Processor's speed constraints* [AHP96, MS93] which are expressed as time intervals between two consecutive events along an instance line. In the MSC-2004 standard [IT04], the delay between any pair of events can be constrained by defining a minimal or maximal bound for the delay between the two events. An interval must define at least one of the two bounds. An absolute time interval can be of the form  $[@1, @3)$  or  $@[1, 3)$ . Figure 3 shows an example of time constraints and measurement. A relative time measurement is used to observe the message duration of the *resolve\_request* call (the time variable *rel1*). The measurement on the duration of the call is subsequently used to restrict the message duration for *resolve\_reply*. The relative time constraint  $(0, 0.7 * rel1]$  allows the message to take at most 70 percent of the time it took to issue the call *resolve\_request* from TC to SUT. In addition, the measurement on the duration of the call is used to constrain the execution of the instance TC: the relative time constraint  $(rel1, 3 * rel1]$  requires that after the output of the *requestNamedAccess* call, it takes at least *rel1* and at most  $3 * rel1$  to get the reply.

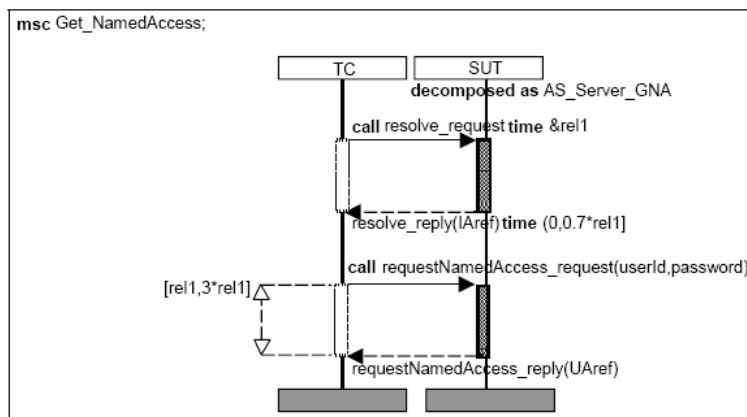


Figure 3: Time Constraints and Measurements [IT04]

- *Timing marks* is a boolean expression on event labels [BRJ96]. For instance the time marker  $a \leq e_i - e_j \leq b$ , where  $e_i$  and  $e_j$  are event labels and  $a$  and  $b$  are real numbers, expresses

that event  $e_j$  must occur within  $[a,b]$  time frame after event  $e_i$ . For basic MSCs, timing marks can be used to describe any timing constraints expressed by timers or delay intervals [LL99a].

Meng-Siew [MS93] used the notion of consecutive events to generalize the message delivery and processor speed delay intervals. He extended the syntax of MSC with precedence edges that connect unrelated events and thus allow the user to provide delay intervals for them. However, these edges may result in a cumbersome and cluttered graph.

Li and Lilius [LL99a] defined the behaviour of an MSC specification as the timed event sequences which are the concatenation of the timed event sequences representing the behaviour of the bMSCs which make up the High-Level MSC specification. Timing constraints are interpreted by *local semantics*: select one path at a time and analyze its timing requirements, independently of other paths that may branch out of the selected one. The authors [LL99a] provided an algorithm to decide about timing consistency.

Alur et al. [AHP96] interpreted timed MSC as partial orders with timing functions that map each pair of events in the partial order to a time interval. In their timed MSC, time constraints can only be imposed on pair of events. They did not consider absolute time constraints at which events occur, and only bMSCs with sending and reception events are addressed. The authors provided also an algorithm for analyzing basic MSCs. Furthermore, the authors proposed an MSC analyzer tool that offers timed analysis based on a semantics that accounts for the queuing strategies in a bMSC and hMSC. Similarly, Ben-Abdallah and Leue [BAL97] used timing delay intervals and timer events to express timing constraints. A MSC is interpreted as traces that are consistent with the partial order of events. They defined a timing assignment that assigns a time stamp to each event in a trace. They also did not consider absolute time constraints. The authors augmented the timing analysis for bMSCs presented in [MS93] and [AHP96] to handle the possibility that a timer is set in a bMSC but not reset nor timeout follows the timer setting in the bMSC. This timing analysis is extended further with branchings and iterations. To address timing consistency, the authors proposed an approach that consists on translating bMSCs into a directed labeled graph, that they call, *temporal constraint graph*. Then this graph is checked to ensure that it didn't contain any cycles with negative cost.

Grabowski et al. [VGO98] defined the semantics of Timed MSCs in terms of Constraint Diagrams (CD) [Die96], a graphical notation for real-time properties stated in the Duration Calculus (DC) [CHR91].

The MSC-2004 [IT04] standard assumes the following time concepts:

- Time progress (i.e. clocking) is equal for all instances in a MSC. Also, all the clock values are equal, i.e. a global clock is assumed.
- All events are instantaneous (i.e. atomic) and do not consume time.
- The time domain can be dense or discrete. It must be a total order with a least element, or origin, of time zero. It must be closed under an addition operation, used to compute time offsets.
- Time constraints can be used to specify the delay between any two events (relative delay), or to specify the time of occurrence of an event (absolute delay). When specifying a relative delay, the time constraint can be an interval with minimal and maximal bounds or a concrete time value. Furthermore, Time constraints can be specified by the use of arbitrary expressions of type *Time*, i.e. referencing parameters, wildcards and dynamic variables.

In the MSC-2004 [IT04] standard, the semantics of a timed MSC are represented by event traces with special time events between normal events. Hence, If there is no time event between two normal events, it means they occur simultaneously. Maigat et al. [LMH00] associate each pair of communication events in MSC with a duration. They propose partial order and (max,+) automaton based semantics and analysis for timed MSC considering HMSC and compositions.

Zheng et al. [ZKH02] provided formal semantics to timed MSCs in terms of *timed labelled partially ordered set (lposet)*. First, they defined the semantics of events as *timed lposets*. Then the semantics of bMSCs, MSCs with structures and hMSCs, are obtained using the operations defined on *timed lposets*. However, their semantics do not cover some MSC standard concepts such as general ordering, instance decomposition, gate and condition. In a related work [ZK02], the authors extended MSCs with a construct, called *instance delay*, to specify repeated MSC scenarios (i.e. specifies how long the scenario takes and the interval between the repetitions). The semantics of this construct is expressed in terms of labelled partially ordered set (*lposets*).

Kim et al. [KC06] proposed *timed high-level message sequence charts (THMSC)* which includes an unambiguous subset of time constraints and timed edges as a new complementary notation. Timed edges are directed time constraints between two consecutive MSCs. The authors claimed that *THMSC* is effective in accurately specifying popular requirement patterns such as watchdog timers and periodic tasks. The formal semantics of *THMSC* is defined using labelled partially ordered set (*lposets*).

#### 4.2. Time-Enriched LSCs

Harel and Marely [HM02] extend LSCs with time information. The authors adopt (1) the approach presented by Alur and Henzinger [AH97], according to which a real-time system can be viewed as a discrete system with clock variables and (2) the synchrony abstraction hypothesis according to which system events consume no real time and time may pass only between events. A single clock object with one property, *Time*, and one operation/method, *Tick*, are used in combination with assignments and conditions to define timing constraints. Time can be stored in time-variables (i.e.  $\text{Time-Variable} := \text{Time}$ ) and compared with time values (for instance the condition:  $\text{Time} > \text{Time-Variable} + \text{Min-Delay}$  is used to specify relative minimal delay whereas  $\text{Time} < \text{Time-Variable} + \text{Max-Delay}$  is used to specify relative maximal delay). The authors distinguish three basic timing constraints:

- *Vertical Delay*: In a single object instance, time is stored upon the occurrence of an event, then the following event is bound by two *hot* conditions defining the minimum and maximum delays of its occurrence.
- *Message Delay*: A message delay is specified similarly, except that the time is stored in one instance line and is checked in another.
- *Timer*: A timer is also specified in the same manner as a vertical constraint, except that the maximal delay condition can be placed arbitrarily far from the place where the time is stored.

Conditions (*hot* and *cold*) can be used to combine timing constraints with conventional constraints to express complex timing constraints.

The timed LSC synchrony hypothesis (i.e. zero-time actions) is implemented in the play-engine simulation tool [HM01]. Indeed, while executing a timed LSC model, the clock keeps ticking and the system waits for external stimuli. When such a stimulus arrives, the execution

freezes the clock and performs the sequence of events that constitutes the system's response to that stimulus. As the sequence is completed, the clock operation is resumed. However, the authors [HM02] have mentioned that the synchrony assumption could be easily dropped by letting the clock continue to tick when events and functions from the model are applied.

In another work by Klose and Wittke [KW01], LSCs are annotated by timers and by delay intervals (both a minimum and a maximum delay) expressing quantitative local liveness properties. However, these intervals are limiting the timing constraints to pairs of events that are either on the same instance line or are connected by a message. The operational semantics of an LSC is defined in terms of a symbolic timed Büchi automaton with unique clocks serving for each constraint. The procedure of deriving an automaton from an LSC is called *unwinding* [KW01].

Table 1: Evaluation of Timed Variants of Message Sequence Charts (1)

Construction Approach	Action/Event Enabling	Durational/Instantaneous Events/Actions	Absolute/Relative Time	Clocks	Urgency	Time Domain
<b>Time Variants of MSC:</b>						
1	TTU MSC-96 [TT96]	delay interval	relative	global	--	?
2	TTU MSC-2004 [TT04]	delay interval	relative/ absolute	global	--	dense/discrete
3	Meng-Siew [MS93]	delay interval	relative	global	--	?
4	Alur et al. [AHP96]	delay interval	relative	global	--	?
5	Li and Lilius [LL99a]	delay interval	relative	global	--	discrete
6	Ben-Abdallah and Lene [BAL97]	delay interval	relative	global	--	?
7	Grabowski et al. [VGO98]	delay interval	relative	global	--	?
8	Matgat and Hérouët [LMH00]	delay interval	relative	?	--	?
9	Zheng et al. [ZKH02]/ Zheng and Khendek [ZK02]	delay interval	?	?	--	?
10	Kim et al. [KC06] (THMSC)	delay interval	relative	global	--	dense/discrete
<b>Time-Enriched LSCs:</b>						
11	Harel and Marely [HM02]	delay interval	absolute	local	--	discrete
12	Klose and Wütke [KW01]	delay interval	absolute	local	--	discrete

Table 2: Evaluation of Timed Variants of Message Sequence Charts (2)

	Construction Approach	Time Representation/ Measurement	Time Expressiveness	Time Analysis	Spec cutability	Exe- cutability	Semantics
	<b>Time Variants of MSC:</b>						
1	ITU MSC-96 [IT96]	point/interval	-	N/A	N/A		denotational
2	ITU MSC-2004 [IT04]	point/interval	+	N/A	N/A		event traces
3	Meng-Siew [MS93]	interval	-	+(bMSC only)	?		?
4	Alur et al. [AHP96]	interval	+	+(bMSC only)	+	(MSC Analyzer)	partial orders
5	Li and Lilius [LL99a]	interval	+	+(timing consistency)	?		local semantics
6	Ben-Abdallah and Leue [BAL97]	interval	+	+(temporal constraint graph)	+	(MSC tool)	event traces
7	Grabowski et al. [VGO98]	interval	-	?	?		constraint diagrams
8	Maigat and Hélotouté [LMH00]	interval	+	+(max,+) automaton	?		partial order automata
9	Zheng et al. [ZKH02]/ Zheng and Khendek [ZK02]	point/interval	+	+	-		timed lposet
10	Kim et al. [KC06] (THMSC)	interval	+	+	?		timed lposet
	<b>Time-Enriched LSCs:</b>						
11	Harel and Marely [HM02]	point/interval	+	?	+	(play engine)	-
12	Klose and Wittke [KW01]	point/interval	+	?	?		timed büchi automata

### 4.3. Timed Annotations in UML

In the following subsections, a survey of timing annotations in UML1.x and UML2.0 diagrams is presented.

#### 4.3.1. UML 1.x

UML 1.x offers nine different diagram types for specifying both structure and behavior of a system. To support real-time modeling, UML 1.x included graphical representation for timing mark to denote event occurrence time, time expression that evaluates to an absolute or relative value of time, and timing constraint which is a semantic statement about the relative or absolute value of time [BRJ96]. However, these added timing constraints are not available in all UML diagrams of the same model and they are generally informal in nature. In UML1.x, there is no time model that describes the way time is progressing. In the following subsections, a survey of timing annotations in UML1.x diagrams is presented.

**UML Timed Sequence Diagrams.** UML sequence diagrams use the drawing rules of message arrows and timing markers to express timing constraints [BRJ96]. A horizontal message arrow indicates the simultaneous occurrence of the send and receive events of the message. A downward slanted message arrow, on the other hand, indicates a required delay between the send and receive events of the message. In addition, within each object outgoing message arrows can be drawn at a single point to indicate the simultaneous sending of a message. Timing markers, boolean expressions placed in braces and attached to the sequence diagram, can also be used to constrain particular events or the whole diagram. These labels (interpreted as time stamps) can be attached at the beginning and the end of a message arrow to specify the minimum or maximum time gap between two marked points in the diagram.

Firley et al. [FHD<sup>+</sup>99] extend UML timed sequence diagrams to express loops. The sequence of messages which is repeated several times is surrounded by a rectangle with the loop condition (i.e. LOOP N TIMES expr) placed at the top or at the bottom of the rectangle. The following convention is used to deal with different occurrences of a labelled event in loops: before a loop,  $a_{first}$  can be used in time constraints, to refer to the time of first occurrence of an event with time stamp  $a$  in the loop. After a loop,  $a_{last}$  refers to the last occurrence of the tagged event in the loop. Within a loop,  $a_{next}$  denotes the time of the event occurrence in the following iteration. The resulting diagrams are translated into observers and implemented in UPPAAL [LPY97]. UPPAAL models are then instrumented to be composed with the observers allowing for formal verification using model checking. However, the presented construction only supports totally ordered sets of events.

In [HHR05] a trace based denotational semantics for timed sequence diagrams is formalized, called the timed STAIRS semantics (Steps to Analyze Interactions with Refinement Semantics). A timed trace is a sequence built from three kinds of events: events for transmission, reception and consumption. Each of these events may have an associated timestamp. The authors [HHR05] claim that these three types of events are introduced to express distinction between black-box and glass-box view of a system. Li and Lilius [LL99b] study timing consistency of both basic UML sequence diagrams and composed sequence diagrams. They showed that the problem of time consistency checking can be reduced to linear programming (i.e. by solving systems of linear inequalities).

**UML Timed Activity Diagrams.** Eshuis et al. [EW01, Esh02] have proposed a formal semantics of UML activity diagrams in terms of Clocked Transition Systems (CTS) [MP96] that is suitable for workflow modeling. The authors [EW01] have proposed two special event labels, *when(text)* and *after(text)*, denoting an absolute and a relative temporal event respectively, where global clock *gc* measures the current time and *text* is an integer expression counting time-units of the global clock and the local clocks. These events are attached to activity diagram transitions. The authors have considered also periodic events, events that are not specified at a single point in time but at a sequence of points. These events are modeled with the *when(cond)* each period construct.

Guelfi and Mammar [GM05] have extended UML activity diagrams with timing constraints. Timing constraints include a time duration attached to each csactivity diagram node and two types of temporal event expressions *After(t)* and *When(t)* similar to the ones proposed in [EW01, Esh02]. The authors have proposed a formal semantics of UML timed-activity diagrams by mapping them to a Clocked Transition System (CTS) [MP96] restricted to integer variables modeling discrete real time aspects. The resulting semantics are translated into PROMELA language for formal verification. One of the limitations of this approach is that external events are not considered.

Li et al. [LMY<sup>+</sup>01] have extended UML activity diagrams by introducing timing constraints. They have introduced a time interval  $[a, b]$  that can be attached to a state *s*. The times *a* and *b* are relative to the moment at which the activity state *s* starts. Assuming that *s* starts at time *c*, then *s* may complete only during the interval  $[c+a, c+b]$  and must complete at the time *c+b* at the latest (i.e. must proceed to the next activity state at the time *c+b* at the latest). Furthermore, the authors have proposed a timing analysis method based on linear programming for UML activity diagrams (containing no loop) and an integer time verification technique for checking more general activity diagrams.

Note: UML profile TURTLE [ACLdSS04], which is discussed in Section 4.3.6, supports temporal operators in activity diagrams.

**UML Timed Statecharts.** Timed Statecharts, proposed by Kesten and Pnueli [KP91], extend the traditional statecharts [Har87] by specifying time limits for the execution of transitions. Their semantics are defined with reference to a dense time domain. Transitions are classified into immediate transitions and timed transitions. Immediate transitions are triggered by inputs, but abstract from time consumption at all. Whenever an immediate transition is enabled, it must be executed before time can proceed. Timed transitions do not depend on inputs. Therefore, they focus on the modeling of time consumption and they are associated with a time interval  $(l, u)$  providing a minimum and a maximum waiting time. The lower bound signifies the minimum time that must be spent in the current state before a transition can be taken, while the possibly infinite upper bound limits the time during which the transition must be taken, if it is to be taken at all. Kesten and Pnueli [KP91] propose a so called *weak time* semantics, i.e., transitions requiring no enabling event and with an associated delay  $\tau$  and timeout  $v$  may be performed (non deterministically) at any time between  $\tau$  and  $v$ . The concept of a *step* is associated with the execution of an immediate transition; a reaction to an event may occur several steps after its generation, but still within the same timestamp. This kind of semantics is based on the fact that every generated event *persists* until the time no longer flow. The time may flow only if all the enabled transitions by that event have been executed.

Peron and Maggiolo-Schettini [PMS94] have considered a version of statecharts with real-time features such as delays and timeouts and allows communicated signals with durations. Al-

though occurrences of actions are related to a dense time domain (i.e., positive rationals), the behaviour of statecharts is forced to be discrete. The authors have also extended the standard notion of reaction by allowing sequences of transitions to be performed instantaneously. Later, in another work, the authors [MSP96] have proposed an approach that assigns a precise duration to transitions instead of time interval [KP91], and enforces a strong time semantics which avoids enabled transitions from being arbitrarily delayed and requires that a non-deterministic choice among transitions is done only if they can be really performed at the same time. Their idea is based on increasing the duration of transitions having null duration, and on decreasing the duration of transitions having nonnull duration, so that the time necessary to perform each chain of transitions remains unchanged.

**UML Profiles.** In addition to the many aspects of UML 1.x that have been criticized (e.g. the metamodel, the usability, the potentially inconsistent diagrams and views, the composition of models, and the insufficient support of error handling, etc.) [HR00, MLLG01], Berkenkotte [Ber03] identified the following four weaknesses related to real-time development:

1. The definition of hardware-software interdependencies: deployment diagrams are too imprecise as they do not provide information on the hardware (except the information that there is hardware at all).
2. The failure to specify timing constraints like deadlines and periods.
3. Communication structures: messages exchange can be specified in various ways (sequence diagrams, collaboration diagrams, etc.), but detailed information like periodicity and protocols cannot be given.
4. Task management policies: UML does not provide mechanisms to describe certain aspects of task management like priorities.

To address some of these weaknesses, UML 1.x has been combined with other techniques like ROOM [SGW94] (discussed in Section 4.5) or SDL [IT02] (discussed in Section 4.4). UML Profiles represent also an alternative to address some of these shortcomings [GK06]. The following subsections list some of the existing UML profiles for real-time modeling:

#### 4.3.2. UML profile for Schedulability, Performance and Time (SPT)

UML profile for Schedulability, Performance and Time (SPT profile) [OMG02] was requested and later adopted by OMG in 2002 to support real-time modeling. UML/SPT is a framework to model resource, time, concurrency, schedulability and performance concepts, in order to support quantitative analysis of UML models.

SPT time domain model identifies the set of time-related concepts and semantics that are supported, directly or indirectly, by this profile. The time domain model is partitioned into the following separate but related groups of concepts [OMG02]:

- Concepts for modeling time and time values. i
- Concepts for modeling events in time and time-related stimuli.
- Concepts for modeling timing mechanisms (clocks, timers). In SPT, clocks were implicitly bound to the physical time.

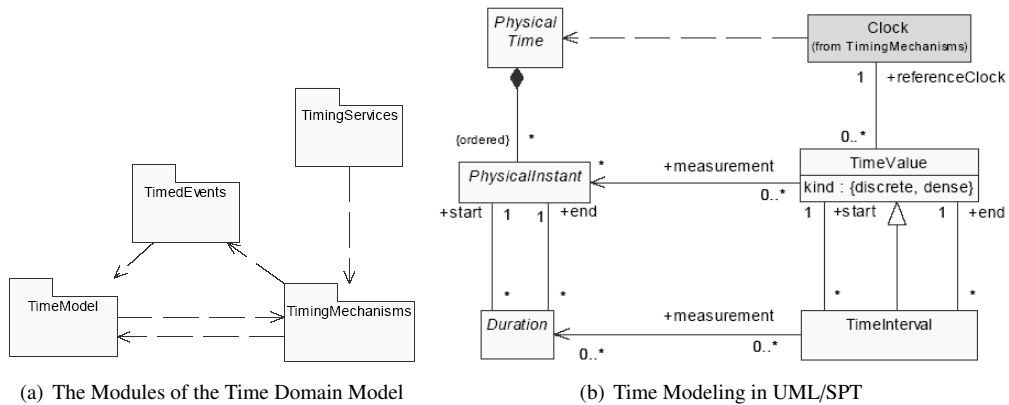


Figure 4: Time in UML/SPT [OMG02]

- Concepts for modeling timing services, such as those found in real-time operating systems.

The underlined concepts are grouped into a set of packages as shown in Figure 4(a).

The sub-profile *<<RTtimeModeling>>* defines a metamodel representing time, as depicted in Figure 4(b), and time-related mechanisms, as illustrated in figures 5, 6 and 7(a). The profile provides a set of stereotypes and associated tagged values that the modeler could apply to UML modeling elements:

- *TimeValue*. There are two ways to specify time values: (1) Use the *RTtime* stereotype to identify model elements that represent time values. The kind of time (discrete or dense) can be specified with an optional tag *RTkind*, which is an enumeration consisting of two elements: *dense* and *discrete*. (2) Use instances of the TVL data type *RTtimeValue* (or its subclasses), which is defined in this profile.
- *TimeInterval*. *RTinterval* stereotype is used to identify instance-based concepts that represent time intervals.
- *TimingMechanism*. The *RTtimingMechanism* stereotype is defined as an abstract stereotype that captures the common characteristics of timers and clocks.
- *Clock*. They are modeled by applying the *RTclock* stereotype. An instance of a clock can be identified using the *RTclockId* tag.
- *Timer*. Timers are modeled by applying *RTtimer* stereotype.
- *TimedAction*. This concept is modeled by applying the *RTaction* stereotype to any model element that specifies an action execution or its specification. This includes action executions, methods, actions, action states, subactivity states, states, and transitions. It can also be applied to model stimuli that take time to arrive at their destination. The start and end times of the action are specified by appropriate tagged values (*RTstart* and *RTend* respectively). Alternatively, they may be tagged with the *RTduration* tag.

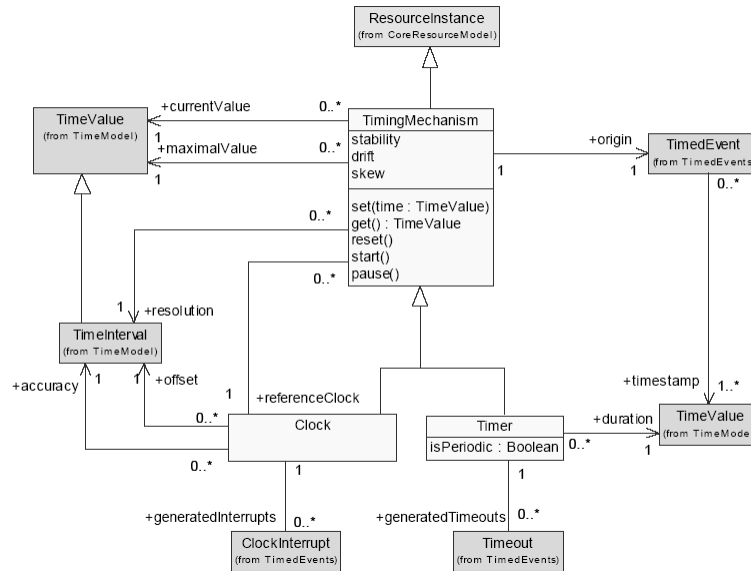


Figure 5: Timing Mechanisms In UML/SPT [OMG02]

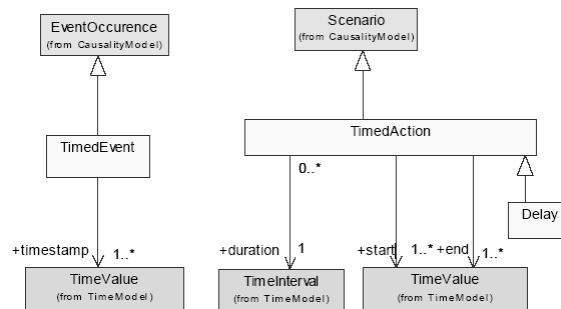


Figure 6: Timed Action and Timed Event Concepts [OMG02]

- *TimedEvent*. This concept is modelled by applying the *RTevent* stereotype to any model element that implies an event occurrence.
- *TimedStimulus*. This concept is useful for modeling any stimulus that has an associated timestamp. This includes invocations of operations, the sending of signals, etc. as well as their descriptors. The stereotype used for this purpose is the *RTstimulus* stereotype which can be attached to stimuli or action executions of actions that generate stimuli.
- *ClockInterrupt*. This is a special type of timed stimulus that is generated by a clock. The stereotype is called *RTclkInterrupt* and it can be applied either to stimuli or messages. The start time (*RTstart*) represents the time of the interrupt.

- *Timeout*. Timeouts are modeled by stimuli or messages that are stereotyped as *RTtimeout*. The start time *RTstart* represents the time of the timeout.
- *Delay*. This is modeled by a model element that is stereotyped as *RTdelay*. It can only have an *RTduration* tag associated with it. Delays can be placed on the same model elements as timed actions.
- *TimeService*. This is represented by stereotype *RTTimeService*. Invocations of the operations of the time service are identified by corresponding stereotypes of ActionExecution or any model element that implies an action execution: *RTnewTimer* and *RTnewClock*.

Figure 7(b) illustrates an example of time annotations in sequence diagrams.

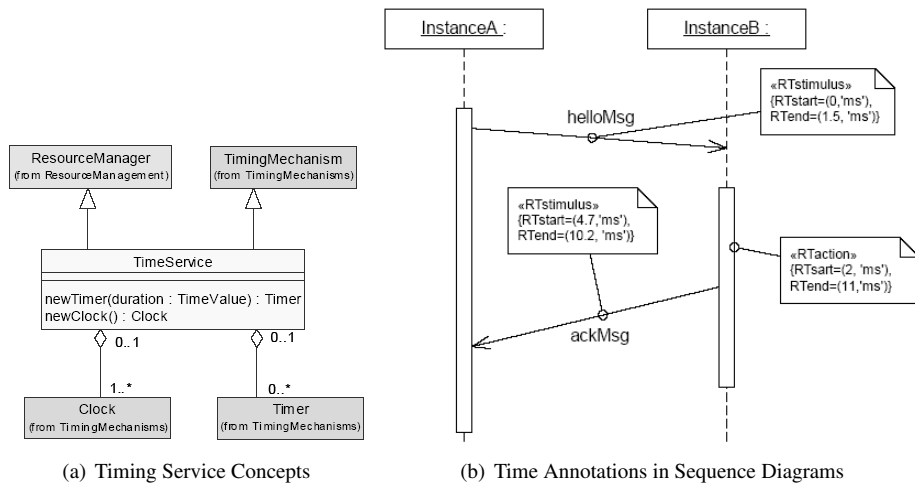


Figure 7: Time annotations in UML/SPT [OMG02]

The SPT profile supports schedulability analysis of UML models by using *«SAprofile»*. SPT schedulability analysis may use modifiers on some parameters, such as: (1) worst-case values (as in, *worst-case execution time*), (2) special parameters of a task, such as its release time, its relative and absolute deadlines and laxity (laxity specifies the type of deadline, hard or soft), and (3) special measures such as blocking time, pre-empted time. In [WP04], Woodside and Petriu have addressed SPT schedulability analysis capabilities and limitations. Other research attempts [KCH01, SKW00] integrate the schedulability theory with object-oriented real-time design.

In addition to SPT, OMG proposes another profile that supports the assessment of non-functional properties of software systems, called the Quality of Service and Fault Tolerance Characteristics and Mechanisms (QoS&FT) [OMG06] profile. QoS&FT allows the user to define a wider variety of QoS requirements and properties. However, QoS&FT requires a tremendous effort to be applied by the final users (software analyst, designer) [BS04]. For a comparative analysis between SPT and QoS&FT, the reader is invited to consult the work by Bernardi and Petriu [BS04].

#### 4.3.3. Modeling and Analysis of Real-Time and Embedded Systems (MARTE)

The OMG has also recently issued a request for proposal (RFP) for a new UML profile for *Modeling and Analysis of Real-Time and Embedded Systems* (MARTE) [OMG07a] in order to upgrade the SPT profile to UML 2 [OMG07b] and to extend its scope with real-time embedded system (RTES) modeling capabilities. MARTE goes beyond the SPT quantitative model of physical time and adopts more general time models. In MARTE, time can be *physical* (used by chronometric clocks), and considered as *dense* or *discrete*, but it can also be *logical* (i.e., bound to any recurrent event), which focus on the ordering of instants, possibly ignoring the physical duration between instants. In a recent work, Peraldi-Frati and Sorel [PFS08], presented a MARTE-based approach to extract temporal information and the implementation characteristics in order to provide a schedulability analysis. Another working line in this context is the one of Espinoza et al. [EDG<sup>+</sup>05] who provided a framework for MARTE by adopting the modeling practices of the SPT and QoS&FT [OMG06], and proposed a domain model for annotating non-functional properties to support temporal verification of UML based models.

Other UML profiles for different quantitative analyses have been proposed in the literature, such as reliability profile [CP04] and dependability analysis profile [BM07].

#### 4.3.4. UML 2.x

UML 2.x [OMG07b] provides two data types: *Time* and *TimeExpression* to express timing constraints. It includes also time related concepts such as *timer* and *clock*. These timing statements can be used either in state diagrams or sequence diagrams as described in the previous sections. Moreover, UML 2.0 [OMG07b] introduces a new diagram called *Timing Diagram* to allow reasoning about time and visualize conditions or state changes over time. Figure 8 illustrates an example of a Timing Diagram.

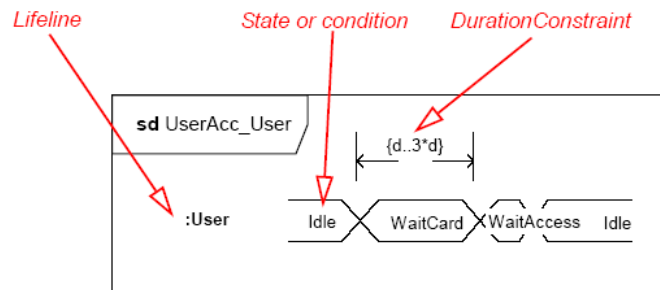


Figure 8: Timing Diagram Example [OMG07b]

Table 3: Evaluation of timed annotations in UML (1)

	Construction Approach	Action/Event Enabling	Durational/Instantaneous Events/Actions	Absolute/Relative Time	Clocks	Urgency	Time Domain
	<b>Unified Modeling Language</b>						
	<b>UML Timed Sequence Diagrams:</b>						
13	Booch et al. [BRJ96]	delay interval	instantaneous	relative	?	--	discrete
14	Firley et al. [FHD <sup>+</sup> 99]	delay interval	instantaneous	relative	?	--	discrete
15	Haugen et al. [HHS05]	delay interval	instantaneous	relative	local	--	
16	Li and Lilius [LL99b]	delay interval	instantaneous	relative	global	--	discrete
	<b>UML Timed Activity Diagrams:</b>						
17	Eshuis and Wieringa [EW01]	?	durational	absolute/relative	global/local	--	dense
18	Guelfi and Mammari [GM05]	?	durational	absolute/relative	global/local	--	dense
19	Li et al. [LMY <sup>+</sup> 01]	?	durational	relative	global	--	discrete
	<b>UML Timed Statecharts:</b>						
20	Kesten and Pnueli [KP91]	initiation and termination of enabling	durational	relative	local	+	dense
21	Peron and Maggiolo-Schettini [PMS94, MSP96]	delay interval	durational	relative	local	+	dense
	<b>UML Profiles:</b>						
22	SPT [OMG02]	?	instantaneous/durational	relative	local, physical	+	dense/discrete
23	MARTE [OMG07a, PFS08, EDG <sup>+</sup> 05]	?	?	relative	local, physical/logical	+	dense/discrete
24	UML 2.x Timing Diagram [OMG07b]	?	Durational	absolute/relative	global	--	dense/discrete /value chain

Table 4: Evaluation of timed annotations in UML (2)

Construction Approach	Time Representation/Measurement	Time Expressiveness	Time Analysis	Spec cutability	Exec-	Semantics
<b>Unified Modeling Language</b>						
<b>UML Timed Sequence Diagrams:</b>						
13	Booch et al. [BRJ96]	point/interval	-	-	-	-
14	Finley et al. [FHD <sup>+</sup> 99]	point/interval	+	observers/ checking	?	TA
15	Haugen et al. [HHR05]	interval	-(basic)	-	-	Timed STAIRS (denotational)
16	Li and Lilius [LL99b]	interval	+	+(timing consistency)	-	local semantics
<b>UML Timed Activity Diagrams:</b>						
17	Eshuis and Wieringa [EW01]	point	+	Model Checking (TCM tool)	-	CTS
18	Guelfi and Mammar [GM05]	point	+	+ Model Checking (PROMELA)	-	CTS
19	Li et al. [LMY <sup>+</sup> 01]	interval	+	+(time consistency)	?	?
<b>UML Timed Statecharts:</b>						
20	Kesten and Pnueli [KP91]	interval	+	-	?	weak time semantics
21	Peron and Maggiolo-Schettini [PMS94, MSP96]	point/interval	+	-	?	strong time semantics
<b>UML Profiles:</b>						
22	SPT [OMG02]	point/interval	+	+ (WCET, resources, scheduling policies)	?	?
23	MARTE [OMG07a, EDG <sup>+</sup> 05]	point/interval	+	Schedulability analysis	Papyrus UML2 editor	?
24	UML 2.x Timing Diagram [OMG07b]	point/interval	-	+	+	?

#### 4.3.5. A UML Profile with the OCL

The Object Constraint Language (OCL) is part of the UML since version 1.3. In UML 1.x versions, OCL was used for specifying invariants attached to classes, pre- and post conditions of operations, and conditions on state transitions. However, it does not provide support for temporal constraints over the dynamic behavior of objects. It is not possible to reference different time instants in a single OCL formula. Only invariant properties can be formalized, which at most include references to attribute values before or after method execution. This leads to several OCL extensions [LMM05, Fla03, CK02, FM02b, FM02a] to address this limitation.

Lavazza et al. [LMM05] have proposed OTL (Object Temporal Logic) as a temporal logic extension to OCL. OTL provides the typical basic temporal operators of temporal logics, i.e., *Always*, *Sometimes*, *Until*, etc. In addition, OTL allows the modeler to reason about time in a quantitative fashion. OTL extends OCL 2.0 standard library by adding three new classes: *Time*, *Duration* and *Interval*. Class *Time* models time instants, which are defined based on the current time taken as the time origin. Class *Duration* models duration of time intervals, i.e., the distance between two time instants. Therefore, a time *Interval* can be defined by its initial time instant and its duration.

Cengarle and Knapp [CK02] extended OCL by satisfaction operators  $@\eta$  to refer to the value in the history of an expression at the instant when event  $\eta$  occurred, as well as the modalities *always* and *sometime*. However, their approach deals with time only from a qualitative viewpoint where no notion of temporal distance between events is provided.

Flake and Mueller [FM02a] proposed a UML profile based on an extension of OCL 2.0 metamodel for the specification of real-time constraints. The formal semantics of this profile is given by means of a mapping to time-annotated temporal logic formulae expressed in CTL, which allows the formal verification of properties. The authors use a discrete time approach.

Sendall and Strohmeier [SS01] proposed an approach to specify concurrent operations through operation schema calculus based on OCL. They have introduced pre- and postcondition assertions, invariants, synchronization on shared resources, signals, and exceptions of system operations written in OCL. The authors have also introduced timing constraints on UML state machines in the context of a restricted form of UML protocol state machines called System Interface Protocol (SIP). A SIP defines the temporal ordering between operations. Five time-based attributes on state transitions are proposed, e.g., (absolute) completion time, duration time, or frequency of state transitions. In a related work, Marcel and de Boer [KdB04] define extension of OCL with a notion of event history that can be used for defining arbitrary constraints on such histories.

#### 4.3.6. Non OMG Profiles

In addition to the aforementioned UML profiles, there are several *unofficial* proposals from the academia considering time modeling.

**OMEGA-RT profile.** This profile, part of the OMEGA project [OME07], aims to provide a concrete UML profile with formal semantics. It is a refinement of the SPT profile. It introduces events based time modeling, *TimedEvent*, where an event is used to represent an instant of state change and allows the expression of duration constraints between occurrences of events [GOO06]. OMEGA-RT profile defines a syntactic classification of events called *Event kinds*. For instance, in a signal exchange, three event kinds can be identified: *send*, *receivesignal* and *acceptsignal* events.

The profile is based on the existence of two basic types, *time* representing points in time *instants* and *duration* representing distances between time points. Sets of instants and durations are expressed by means of predicates. These predicates are formalized using OCL-like expressions. OMEGA-RT profile define the following duration patterns [GOO06]:

- execution time, execution delay, client response time, server response time, transmission delay which are associated with actions.
- reactivity and period which are associated with a trigger
- transmission delay associated with a communication channel
- lifetime associated with an object, and many more.

For requirements involving conditions, which are more complex than the distance between two events, OMEGA-RT introduces the *observer* formalism, defined by the stereotype class of state machine ( $\ll\text{observer}\gg$ ). An observer is an object which executes synchronously with a system and monitors its state and the events that are occurring. Note that the OMEGA-RT event is different from the UML event, which poses a compliance issue.

**TURTLE profile.** TURTLE (Timed UML and RT-LOTOS Environment) [ACLdSS04] is a real-time UML1.5 [OMG03] compliant profile with formal semantics given in terms of RT-LOTOS. TURTLE profile extends class/object diagrams and activity diagrams of UML1.5 [OMG03]. TURTLE class diagram consists of *TClasses* having special attributes called Gates. Gates are used by *TClass* instances, *TInstances*, to communicate and are specialized into *InGate* and *OutGate*. In addition, TURTLE introduces stereotypes called composition operators that are used to explicitly express parallelism, synchronization, and sequence relationships between *TClasses*. In TURTLE profile, activity diagrams implement the behaviour of a *TClass*. These activity diagrams use logical and temporal operators that allow expressing synchronization on gates with data exchange. For real time modeling TURTLE offers the following temporal operators: deterministic delay, nondeterministic delay, timelimited offer, and time capture operator (see Figure 9). Time intervals are expressed by combining the deterministic and nondeterministic delays.

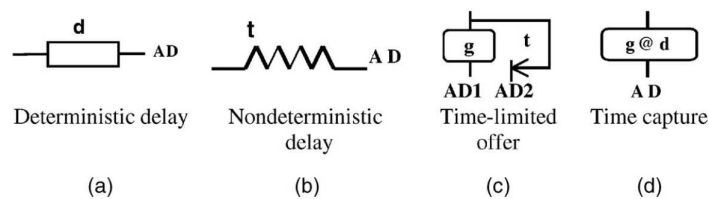


Figure 9: TURTLE Temporal Operators [ACLdSS04]

TURTLE has been extended to include UML component and deployment diagrams. The resulting profile is called TURTLE-P [ALSS<sup>+</sup>06], which addresses the concrete description of communication architectures including quality of service parameters (delay, jitter, etc.). TURTLE-P allows the evaluation and formal validation of UML components and deployment diagrams. TURTLE is supported by TTool [TTo07]. TTool is linked to RTL [RL07b], the RT-LOTOS validation toolkit developed at CNRS, and to CADP [RL07a], a formal validation toolkit developed at INRIA.

**The European EAST-EEA.** The European EAST-EEA (Electronic Architecture and Software Technology – Embedded Electronic Architecture) [EE04] is an ITEA (Information Technology for European Advancement) project [ITE04]. It provides a development process and automotive-specific constructs for the design of embedded electronic applications. It provides the concepts of *Triggers, Period, Events, End to End Delay, Physical Unit* that can be applied to any behavioral EAST elements. In practice, some of these concepts, such as the event triggering, make the timing analysis very complex. In the EAST-ADL (Architecture Description Language) document, it is recommended to use event triggering carefully or even to avoid it [AMPF07]. EAST-EEA is compliant with UML2.0.

Roubtsova et al. [EERdR01] define a UML profile with stereotyped classes for dense time as well as parameterized specification templates for deadlines, counters, and state sequences. Each of these templates has a structural-equivalent dense-time temporal logics formula in Timed Computation Tree Logic (TCTL). The authors [EERdR01], though, do not use OCL for constraint specification in their formal approach.

#### 4.3.7. IF language

The IF language [BFG<sup>+</sup>99, BFG<sup>+</sup>00] and the associated toolset developed at VERIMAG were developed for modeling and validating distributed systems that can manipulate complex data and both involve dynamic aspects and real time constraints. The IF language describes the operational semantics of higher level formalisms such as UML or SDL, and is also used as a format for inter-connecting model-based tools. An IF description defines the structure of a system and the behavior of its components. A system is composed of a set of communicating processes that run in parallel. IF provides support for real time constraints expressed using clock variables and guard conditions on them. The values of such variables increase with time. The underlying semantics is based on finite timed automata with urgency [AD94, BS00]. The IF language and tool-set [OGO06] translates timed UML models into timed automata in which UML level concepts are mapped into more primitive concepts. IF language format is used for the mapping and existing model-checking tools can be used for validation.

Table 5: Evaluation of OCL and non OMG UML Profiles (1)

	<b>Construction Approach</b>	Action/Event Enabling	Durational/ Instantaneous Events/Actions	Absolute/Relative Time	Clocks	Urgency	Time Domain
	<b>OCL:</b>						
25	Lavazza et al. (OTL)[LMM05]	?	instantaneous	absolute	global	--	dense/discrete
26	Cengarle and Knapp [CK02]	?	instantaneous	absolute	global	--	dense/discrete
27	Flake and Mueller [FM02a])	?	?	absolute	?	--	discrete
28	Sendall and Strohmeier [SS01]	?	?	absolute	?	---	dense
	<b>non OMG Profiles:</b>						
29	OMEGA-RT [OME07]	?	instantaneous	relative	local, physical	+	dense/discrete
30	TURTLE [ACLDSS04] and TURTLE-P [ALSS+06]	?	instantaneous	relative	local	+	dense/discrete
31	EAST-EEA [EE04]	?	?	?	physical	--	?
32	Roubisova et al. [EERdr01]	?	?	?	?	--	dense
33	IF language [BFG+99] [BFG+00]	?	instantaneous	relative	local	+(TA)	dense

Table 6: Evaluation of OCL and non OMG UML Profiles (2)

	Construction Approach	Time Representation/ Measurement	Time Expressiveness	Time Analysis	Spec cutability	Exe- cutability	Semantics
	<b>OCL:</b>						
25	Lavazza et al. [LMM05]	point/interval	+ (OTL)	?	N/A		?
26	Cengarle and Knapp [CK02]	point	+ (OCL + temporal operators)	-	-		trace semantics
27	Flake and Mueller [FM02a])	interval	+ (OCL consistent)	Formal verification	N/A		Clocked CTL
28	Sendall and Strohmeier [SS01]	?	+ (OCL consistent)	-	N/A		-
	<b>non OMG Profiles:</b>						
29	OMEGA-RT [OME07]	point/interval	+	+	?		
30	TURTLE [ACLS04] and TURTLE-P [ALSS <sup>+</sup> 06]	point/interval	+	+	+		RT-LOTOS
31	EAST-EEA [EE04]	point/interval	-	?	?		?
32	Roubtsova et al. [EERdr01]	point/interval	+ (stereotyped classes)	?	?		TCITL
33	IF language [BFG <sup>+</sup> 99] [BFG <sup>+</sup> 00]	point/interval	+	+(model checking)	+		TA

#### 4.4. Time in SDL

The standard semantics of SDL, as presented in Z.100 [IT02] (expressed by means of Abstract State Machines), is very abstract in the sense that it makes no assumptions on time consumption and progress. Tasks may take an indeterminate amount of time to execute, and a process may stay an indeterminate amount of time in a certain state before taking the next fireable transition. Control over these durations is left to particular implementations by tool vendor (depends on application domain, implementation architecture, or purpose of simulation). However, SDL has some features that can be used to model aspects of timed systems, such as global system time (represented by a system clock (*now*) and allows to measure durations throughout the system by means of appropriate time stamps). SDL supports two time-related data types: *Time* and *Duration*; *Time* should be used to denote a point in time, while *Duration* should be used to denote a time interval. System clock (*now*) is external to the specification. For example the system clock cannot be reset within the specification, nor does it progress in an orderly fashion. Rather, the only means for any form of control over the system clock is through the usage of timers.

SDL allows the description of time dependent functional behaviours by means of *timers*, *enabling conditions* and *continuous signals*. A timer expires when some delay has been exceeded, resulting in an input signal being placed in the input queue of the associated process. However, there is no guarantee when the signal will be consumed [BGM<sup>+</sup>01]. SDL timer primitives are *set* and *reset* operations, the *active* function (which gives the state of a timer) and timeouts that are always transmitted in the form of asynchronous signals. An enabling condition, referring to the system time *now*, can be attached to an input signal. A continuous signal can also refer to *now*, where the intention is that when some time constraint is satisfied in a state, the behaviour of the process can evolve without environmental interaction (i.e. without an input signal). However, these two constructs do not allow the specification of transitions which are taken at a specific point of time (or within a specific time interval), as there is no notion of urgency in SDL. The current SDL semantics [IT02] treats all transitions as lazy since it places no constraints on time progress. Most SDL tools however implement an eager semantics where transitions are fired as soon as they are enabled without letting time progress. Part of the European IST project INTERVAL (1999-2002), Bozga et al. [BGM<sup>+</sup>01] propose a more flexible time semantics for SDL based on timed automata with urgencies [BST98].

Real-time distributed systems lack the notion of a global system clock, and thus global time. Graf [Gra02] proposed to introduce the notion of local time (defined by a drift and/or offset with respect to the global time *now*). She suggested that a defined relationship between the external reference time (*now*) and local time must always exist.

The QSDL [DHHMC95] defined an extension of SDL with probabilistic execution time constraints attached with tasks and a notation for some minimal deployment information. QSDL defined time durations (deterministic and probabilistic), timed transitions (using an action called *request*) and timed states (using the *awake*-construct). The resulting models are then fed into the tool QUEST, which transforms a QSDL-specification into an automatically assessable model (called an evaluator) for performance and time related verification.

#### 4.5. Real-time Object Oriented Modeling (ROOM)

ROOM (Real-Time Object-Oriented Modeling), originally introduced in [SGW94], is a methodology that was developed primarily for distributed real-time systems based on the object paradigm. Modeling of systems with ROOM is performed by modeling actors (the central component of ROOM), which are encapsulated concurrent objects, communicating via point-to-point links.

The behavior of an actor is represented by an extended state machine called a *ROOMChart*, based on Harel statechart formalism [Har87]. Inter-actor communication is performed exclusively by sending and receiving messages via interface objects called *ports*. A message is a tuple consisting of a signal name, a message body (i.e., data associated with the message), and an associated message priority. The original ROOM [SGW94] does not provide any mechanisms to constrain the behaviour of actors (for instance to specify and enforce timing constraints). Instead system behaviour may be derived using Message Sequence Charts, which can be annotated with timing constraints [BAL97, SFR97]. The ROOM developers use the term *transaction*, to describe end-to-end computations on which timing constraints such as periodicity and deadlines may be specified. MSCs are used to express: (1) activation periods of methods (which represents either the inter-arrival time of the periodic timer, or a minimum inter-arrival time for aperiodically triggered transactions) and (2) end-to-end deadlines on sequences of message invocations (which represents the response time of the transaction). Using these two types of timing constraints and a few design guidelines, the authors in [SFR97] show how scheduling theory can be applied to ROOM models.

ROOM concepts were supported by a commercial CASE tool called *ObjecTime* (ObjecTime Ltd., Kanata, Ontario, Canada). They have been also incorporated into the CASE tool *Rational Rose Real-Time (RoseRT)* in the form of a UML profile, commonly called *UML-RT*.

#### 4.6. Visual Timed Event Scenarios (VTS)

Alfonso et al. [BKO05] introduced VTS, a visual language to define event-based requirements such as freshness, bounded response, event correlation, etc. The underlying language is based on partial orders and supports real-time constraints in a dense time domain. Figure 10 summarizes the VTS graphical notation.

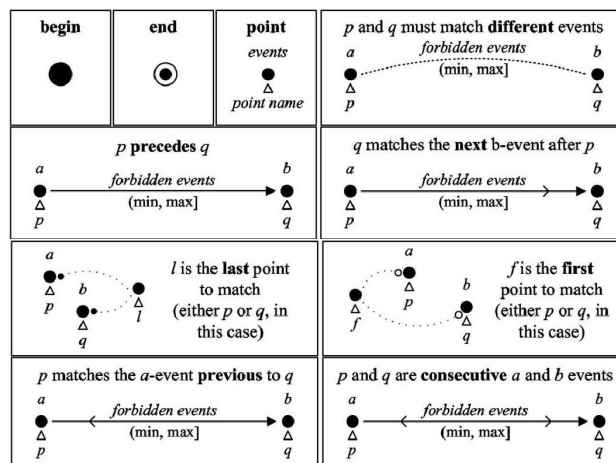


Figure 10: VTS Graphical Notation [BKO05]

The authors [BKO05] provided a declarative (denotational) semantics of VTS, where a set of traces are assigned to each VTS scenario and labelled points represent events in the traces. Points that are not labelled are called *instants*. They represent moments in the execution not necessarily

associated with an event. The resulting semantics are not executable. VTS is supported by a tool that translates visually specified scenarios (the ones violating the requirements) into observer timed automata. The resulting automata can now be composed as a model for further analysis, in order to check if the stated scenarios satisfy all conditions (e.g. using model checkers like UPPAAL [LPY97] and Kronos). However, describing graphically all possible scenarios that violate a given requirement is an error prone activity and the resulting set of scenarios may be incomplete. Furthermore, unlike other timed notations, such as LSC [HM02] and timed sequence charts [HHR05], VTS does not use the *timer* concept and it abstracts from the instances that perform events (e.g. message exchange).

#### 4.7. Property Sequence Chart (PSC)

Property Sequence Chart (PSC), initially introduced in [AIP07], is a scenario-based visual language that extends the graphical notation of a subset of the UML 2.0 Interaction Sequence Diagrams. Autili et al. [AIP07] have provided a comparison between PSC, UML 2.0 Interaction Sequence Diagrams and MSC, based on the existence of the following features: undesired/mandatory/provisional message, strict/weak sequencing, restrictions on intraMSGs (messages possibly exchanged in the past, present and future), all of messages but one, simultaneous messages, interaction construct and parallel operator.

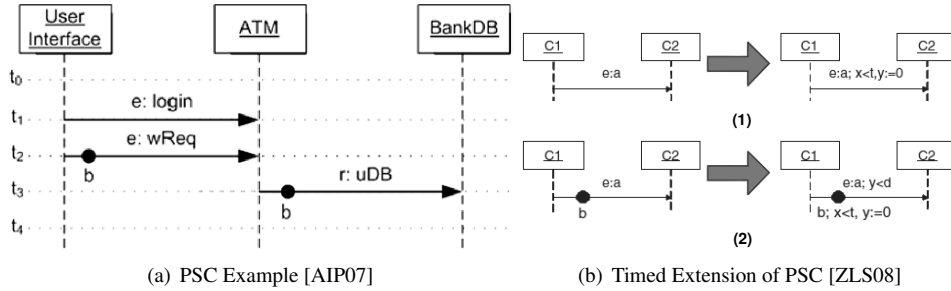


Figure 11: Property Sequence Chart (PSC)

Although PSC provides a simple and user friendly formalism for specifying temporal properties, time support remains weak since the language does not allow the description of timing constraints. As shown in Figure 11(a), time is absolute and the only offered time notation consists of a set of horizontal dotted lines  $t_0, \dots, t_n$  called *time-lines*. No timing constraints are defined in order to be able to define a lower and an upper bound between two subsequent messages on one instance. The language is supported by a tool, called *CHARMY* [PIM08], that can be used to translate specified properties into a test automaton (i.e., Büchi automaton).

Zhang et al. [ZLS08] have extended PSC into Timed PSC (TPSC). As shown in Figure 11(b)(1), a regular message  $e:a$  is extended as  $e:a; x<t, y:=0$ , which means  $a$  may happen before  $t$ , but if it does not happen, the system has no error. A new clock  $y$  is initialized after the regular message  $a$  is exchanged. Required messages with restricted time constraints must be exchanged. Figure 11(b)(2) shows a regular message with past Boolean constraints, which means past constraint  $b$  may hold before  $t$ , then, a regular message  $a$  may be exchanged within next  $d$  time units. However, if  $b$  does not hold and  $a$  is not exchanged within the desired time, it will

be no error. The authors [ZLS08] have provided the semantics of TPSC in terms of timed Büchi automaton. However, neither formal verification nor time analysis were considered.

#### 4.8. Real-time Graphical Interval Logic (RTGIL)

Real-time graphical interval logic (RTGIL) [MRK<sup>+</sup>97], and its corresponding textual representation, real-time future interval logic (RTFIL), are real-time extensions to graphical interval logic (GIL) [RMSM<sup>+</sup>96], and its textual representation, future interval logic (FIL), respectively. RTGIL is a propositional linear-time temporal logic, interpreted over dense time. In RTGIL, a time line is used to show the progression of a computation. Intervals can be constructed on this time line; an interval is represented by a segment of the time line delimited by two states and is left-closed and right-open. Intervals are constructed using search patterns with associated target formulae. A search locates the first state in the future from the current position on the time line where the target formula holds (which might be the current state if the formula holds there). Formulae are read from top to bottom and from left to right, and can be combined using standard logical infix operators. Initial properties (Figure 12(a)) as well as henceforth or eventuality properties can be assigned to an interval. Figure 12(a) asserts that  $h$  holds at the first state if the interval that begins with the first state at which  $f$  holds and ends just prior to the next state at which  $g$  holds. The only real-time operator supported by RTGIL is the  $len$  predicate, for example ( $len(d, D]$  in Figure 12(b) implies that the duration of the interval, if it can be constructed, is greater than  $d$  time units and less than or equal to  $D$  time units ( $d$  and  $D$  represent non negative rational constants, where  $D$  can also be  $\infty$ ).

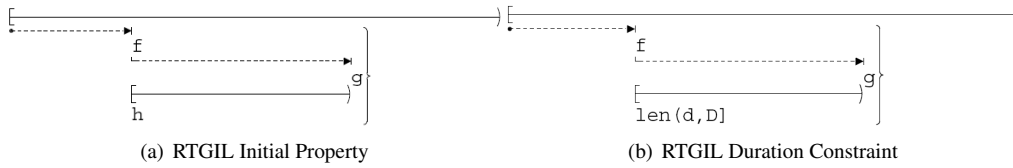


Figure 12: RTGIL Examples [MRK<sup>+</sup>97]

RTGIL is supported by the RTGIL environment [MRK<sup>+</sup>97], which includes a graphical editor, an automated theorem prover, and a data base and proof manager component. Because the RTGIL environment is a homogeneous analysis tool, a model and its correctness properties are both specified in terms of RTGIL formulae.

#### 4.9. Timeline Notation

A timeline [SHE01] is represented by a wide horizontal bar, with time progressing from left to right. The vertical bars, called *marks*, mark the interesting event occurrences, ordered in time. Timeline defines two types of system events: regular events (denoted by the letter  $e$ ) and required events (denoted by the letter  $r$ ). The events can be generated anywhere in the system, by any one of many concurrent processes in the distributed system. Therefore, no fixed time-interval can be assumed between subsequent marks (there is no hidden assumption of a *global clock*). Timeline notation allows for describing constraints represented as black horizontal lines positioned beneath the timeline bar. These constraints are used to specify the occurrence of particular events over certain intervals. Figure 13 describes the fact that when the system must

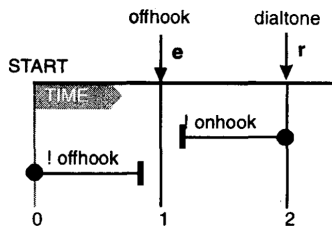


Figure 13: Timeline example [SHE01]

respond to an offhook by providing dialtone, the constraint !onhook must be placed within the interval between the ofhook and the dialtone event.

Timeline notation is supported by a graphical tool called TimeEdt [SHE01]. It was developed to capture series of events and required system responses. These complex chains are placed on a timeline and may be converted into a test automaton, that can be used directly by a logic model checker, or for traditional test-sequence generation. Even if intuitive, TimeEdt do not feature partial ordering of events and does not support complex timing constraints.

#### 4.10. Regular Timing Diagrams (RTD)

Regular Timing Diagrams [AEN99, AEKN01] are a known notation in the context of hardware design. RTDs describe, over a finite time period, changes of signal values, and precedence and timing dependencies between such events, such as *signal a rises within 5 time units of signal b falling* and *signal b is low when signal a rises*. Such events can be causally constrained and time-constrained by a number of ticks of a given clock where the time intervals are specified by constants, ensuring that the diagram defines a regular language.

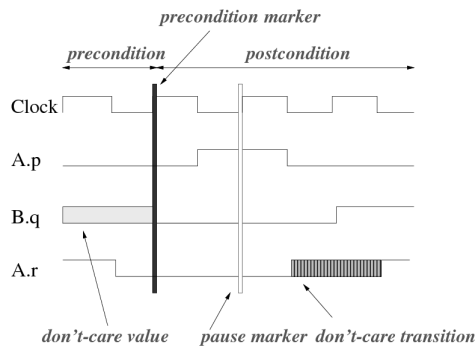


Figure 14: Synchronous Regular Timing Diagram [AEKN01]

A RTD may be either asynchronous or synchronous. A synchronous diagram (SRTD) includes one or more clocks with fixed periods and ensures that the time interval between any pair of events is determined up to the clock period (see Figure 14). Any change in the signal value must occur at either the rising edge or falling edge of the clock waveform (which is between 0 and 1). The ordering between events is in general partial; such RTDs are considered as ambiguous. An unambiguous RTD has a total ordering on events.

#### 4.11. Action Diagrams (Timing Diagrams)

An action diagram (AD) [Kho96, KC98] specifies in a declarative manner the interface behavior of a system. The specification comprises the interface behavior of the system itself (its commitments), as well as the assumptions that the system makes on its environment. Both commitments and assumptions are described by ports, actions, and timing constraints. Ports are abstractions of the logic signals used by the system to communicate with its environment. A direction (in or out) and a sequence of actions are associated with every port. Actions occur instantaneously; they represent punctual changes of the logic values of these signals. An action  $a_k$  has a time stamp variable denoted by  $t(a_k)$  which is a finite real value (dense time model).

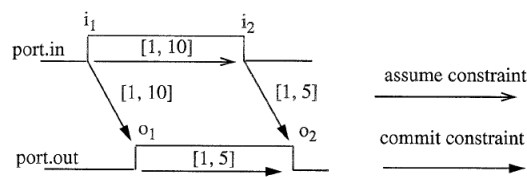


Figure 15: Example of Action Diagram [KC98]

In the graphical representation of action diagrams, an action is represented by a short vertical bar (e.g., Figure 15). Actions on the same port are horizontally aligned. The action sequence of a port is shown in left-to-right order. A constraint  $(a_i, a_j, \pi)$  is represented by an arrow labeled with the interval  $\pi$  and pointing from  $a_i$  to  $a_j$ . The constraint arrowhead is hollow (filled) to represent assume (commit) constraints. Khordoc and Cerny [KC98] have provided algorithms to check for AD consistency, compatibility, liveness and causality.

#### 4.12. Timed Behavior Trees

In a recent work, Grunke et al. [GWC07] extended the Behaviour Trees (BT) notation, initially introduced in [Dro03], to include timing constraints. A timed BT model is equipped with a number of clocks which evaluate to a real number. All clocks progress simultaneously. A clock can be reset to zero or can constitute a guard on a transition or an invariant on a location. BT nodes [Dro03] are extended with three additional slots (see Figure 16): a guard  $G$  over clock values, a reset  $R$  of clocks, an invariant  $I$  over clocks. Nodes in a timed Behavior Tree describe transitions from one location to the next as they describe a state change, a guard or message passing.

Later, Colvin et al. [CGW08] have defined the semantics of timed BTs using timed automata where each language concept has an equivalent automaton or a network of automata. The resulting semantics can be used as an input for the model checker UPPAAL. Furthermore, Colvin et al. [CGW08] have introduced a novel approach for supporting Failure Mode and Effects Analysis (FMEA) for time-critical systems.

#### 4.13. Somé's Scenarios.

Somé et al. [SDV95, SDV96] represent timed scenarios with structured text, but also with a formal interpretation where preconditions, triggers, sequence of actions, reactions and delays are specified [SDV96]. Scenarios are interpreted as timed sequences of events, which make them appropriate for real-time systems. External events represent interactions between components,

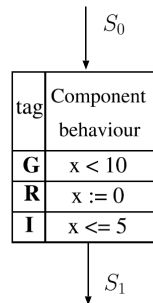


Figure 16: Timed BT Node [GWC07]

including actors, whereas actions can be internal. The time of occurrence of operations can be constrained by interaction *initial delays* and *timeouts* and scenario *timeouts*. An interaction *initial delay* specifies a minimal, a maximal or an exact amount of time that must pass between the interaction first operation, and the last operation of the interaction preceding it. Specifications described in this notation can be implemented in the Use Case Editor tool (UCEd)[Som04] and can be translated into a timed automata specification.

#### 4.14. Timed Use Case Maps

The Use Case Maps language (UCM), part of the ITU-T standard User Requirements Notation (URN) Z.151 [IT08], is a high level scenario based modeling technique that can be used to capture and integrate functional requirements in terms of causal scenarios representing behavioral aspects at a high level of abstraction.

Use Case Maps notation has been extended with time constraints (*timed Use Case Maps* [HRD06]). The following summarizes the adopted time criteria:

- *Initiation and termination of enabling* [BG06] represents a flexible and suitable choice for UCM level of abstraction. Both a lower and upper bound may be imposed on the enabling of a responsibility. A responsibility may be enabled any time between *minDL* and *maxDL* time units after the completion of its predecessor. A such delay is introduced in order to describe, for instance, situations of queueing delay or when the resources needed to execute a responsibility are not immediately available.
- *Durational semantic model*: Time is mainly consumed by responsibilities. A responsibility may be associated with *minDur* and *maxDur* denoting respectively its best and worst case execution times. UCM control constructs such as OR-Forks (involving condition evaluation) may take some time to complete.
- A global and centralized clock for measuring and increasing time globally over the system is used (i.e. *MasterClock (MClock)*).
- Either dense or discrete time model can be used.
- Both *relative* and *absolute* time models are considered. Relative time is used to define the duration of responsibilities and their incurred delay. Absolute time is used to track the value

of the master clock (i.e.  $MClock$ ). It is used in start points to record the scenario starting time and to define responsibilities' deadlines.

- *Urgency*: A responsibility is considered as *urgent* when enabled immediately after the execution of its predecessor (i.e.  $minDL = maxDL = 0$ ). Alternatively, it is considered as *delayable*. All UCM control constructs (i.e. OR-Fork, OR-Join, AND-Fork, etc.) are considered as urgent once enabled. Transitions are processed as soon as they are enabled, allowing for a maximal progress. Figure 17 illustrates a delayable responsibility  $R$  with an estimated delay within  $[1,2]$ , an execution interval of  $[10,20]$  and a deadline of 25 (i.e.,  $MClock = 25$ ). The start point  $S$  is triggered at  $MClock = 0$ .

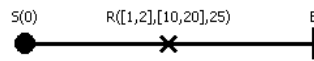


Figure 17: Delayable Responsibility

Three formalization approaches were provided for timed UCM language:

- Clocked Transition Systems (CTS) [MP96] based semantics: Based on a discrete time model, Hassine et al. [HRD06] defined two step semantics (i.e. two sets of transition rules) for timed UCM models, to cover both interleaving and true concurrency models.
- Abstract State Machine (ASM) [Gur88] based semantics: Based on a discrete time model, the untimed ASM semantics, introduced in [HRD05b, HRD05a], have been extended to cover the Timed Use Case Maps language [Has08]. The proposed approach has two benefits. First, it is relatively cheap to implement since it is built upon the untimed ASM operational semantics. Second, it provides an environment (AsmL based) to simulate (one shot or step-by-step simulation) and to capture various aspects of a system run (e.g. execution time, executed constructs, components, etc.) in one single timed trace. However, this approach does not support true concurrency model semantics.
- Timed Automata (TA) [AD94] based semantics: Based on a dense time model, the authors in [HRD07] formalized timed UCM in terms of timed automaton allowing for formal verification using UPPAAL model checker [LPY97].

For a detailed explanation the reader is invited to consult [Has08].

#### 4.15. Timed Petri-Nets

Time in Petri-Nets can be associated with:

- Tokens: Each token is associated with a time-stamp  $\theta$  that indicates when the token is available to fire a transition (Figure 18(a)).
- Places: Timed Place Petri Nets (TPPN): Each place  $p$  is associated with a delay attribute  $t$ . Tokens generated in  $p$  only become available to fire a transition after the delay  $t$  has elapsed (Figure 18(c)).
- Arcs: Each arc is associated with a traveling delay  $t$ . Tokens are available for firing only when they reach the transition (Figure 18(b)).

- Transitions: Timed Transition Petri Net (TTPN): Each transition represents an activity which starts when the transition is enabled and terminates when the transition is fired (Figure 18(d)). The two main extensions of Petri Nets for handling time are *Time Petri Nets* (TPNs) [Mer74] and *Timed Petri Nets* (TdpNs) [Ram74].

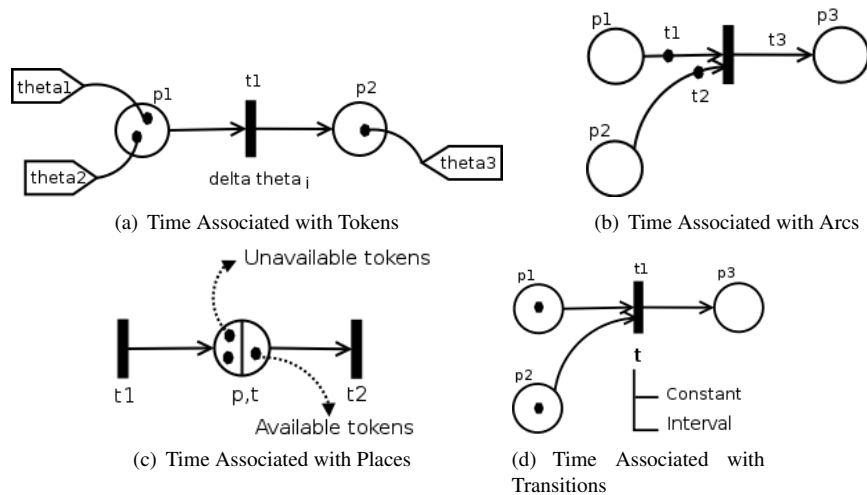


Figure 18: Timed Petri-Nets [Mer74]

Timed Petri Nets (TdpN) [Ram74], also called timed-arc Petri Nets, associate with each arc an interval (or bag of intervals). In TdpNs, each token has an age. This age is initially set to a value belonging to the interval of the arc which has produced it or set to zero if it belongs to the initial marking. Afterwards, ages of tokens evolve synchronously with time. A transition may be fired if tokens with age belonging to the intervals of its input arcs may be found in the current configuration.

TPNs [Mer74] associate with each transition a time interval. A transition can be fired if its enabling duration lies in its interval and time can elapse only if it does not disable some transition. Firing of an enabled transition may depend on other enabled transitions even if they do not share any input or output place.

The concept of associating random time durations to transitions was first investigated by Natkin [Nat85] and this was the origin for the emergence of stochastic Petri Nets and their extensions. For an extensive survey of the semantics of Petri Nets with time, the reader is invited to consult the work of Cerone et al. [CMS99].

Table 7: Time in SDL and other notations (1)

	Construction Approach	Action/Event Enabling	Durational/ Instantaneous Events/Actions	Absolute/ Rela- tive Time	Clocks	Urgency	Time Domain
	<b>SDL:</b>						
34	SDL [IT02]	simple	N/A	absolute	physical, global	-	dense
35	Graf [Gra02]	initiation and termina- tion of enabling	instantaneous	absolute	local	+	dense
36	QSDL [DHHMC95]	initiation and termina- tion of enabling	durational	absolute	global	+	dense
	<b>Other Timed Scenario Notations:</b>						
37	Real-time Object Ori- ented Modeling(ROOM) [SGW94, SFR97]	simple	instantaneous	relative	global	?	?
38	Visual Timed Event Scenarios (VTS) [BKO05]	simple	instantaneous	relative	N/A	N/A	N/A
39	Timed Property Sequence Chart (PSC) [AIP07, ZLS08]	simple	instantaneous	absolute	--	--	--
40	Real-time Graphical Interval Logic (RTGIL) [MRK <sup>+</sup> 97]	simple	instantaneous	relative	--	N/A	dense
41	Time Line [SHE01]	simple	durational	absolute	--	N/A	?
42	Regular Timing Diagrams (RTD) [AEN99]	simple	durational	absolute	global	--	discrete
43	Action diagram (AD) [Kho96, KC98]	simple	instantaneous	relative	global	--	dense
44	Timed Behavior Trees [GWC07]	initiation and termina- tion of enabling	instantaneous	relative	local	+	dense
45	Some Scenarios [SDV95]	simple	instantaneous	absolute/ rela- tive	?	--	?
46	Timed Use Case Maps [HRD06, HRD07, Has08]	initiation and termina- tion of enabling	durational	absolute/ rela- tive	global/ lo- cal	+	dense/discrete
47	Timed Petri Nets [Mer74, Ram74, CMS99]	initiation and termina- tion of enabling	durational	absolute	global		dense

Table 8: Time in SDL and other notations (2)

Construction Approach	Time Representation/Measurement	Time Expressiveness	Time Analysis	Spec cutability	Exec-	Semantics
<b>SDL:</b>						
34 SDL standard [TT02]	interval	-	N/A	N/A		ASM
35 Graf [Gra02]	interval	+	?	?		Transition Urgencies
36 QSDL [DHHMC95]	point/interval	+	+ (timed validation)	+ (QUEST)		?
<b>Other Timed Scenario Notations:</b>						
37 Real-time Object Oriented Modeling(ROOM) [SGW94, SFR97]	point/interval	+	schedulability analysis	+ (UML-RT, object time)		EFSM
38 Visual Timed Event Scenarios (VTS) [BKO05]	point/interval	--	TA observers	--		denotational semantics
39 Timed Property Sequence Chart (PSC) [AIP07, ZLS08]	point	--	--	--		buchi automata
40 Real-time Graphical Interval Logic (RTGIL) [MRK+97]	point/interval	+	theorem prover	N/A		--
41 Time Line[SHE01]	-	-	--	-		-
42 Regular Timing Diagrams (RTD) [AEN99]	interval	--	--	--		--
43 Action diagram (AD) [Kho96, KC98]	interval	-	Consistency, compatibility, liveness and causality checkings	+		operational
44 Timed Behavior Trees [GWC07]	point/interval	+	+ (model checking)	?		TA
45 Some Scenarios [SDV95]	interval	+	+ (TA)	?		TA
46 Timed Use Case Maps [HRD06, Has08, HRD07]	point/interval	+	model checking	+ (ASML, UP-PAAL)		TA/CTS/ASM
47 Timed Petri Nets [Mer74, Ram74, CMS99]	interval	+	+	+		Formal

## 5. Evaluation Summary

In this section, we summarize the results of our state-of-the-art survey of timed scenario-based approaches and discuss observations made as part of the survey. For the comparison of the timed scenario-based modeling approaches, we have introduced eleven time-related criteria which can be classified in two main categories:

- *Time characteristics.* This set represents criteria dealing with time design choices such as *time domain*, *time representation* and *Clock type*.
- *Usability of the notation:* This set includes criteria that impact the deployment of the notation, such as *Time expressiveness*, *executability* (which includes tool support), *Time Semantics* and *Time Analysis*.

The surveyed notations and construction approaches differ with respect to their expressive power, goals and application domain. Timed scenario notations based on messages exchanged between communicating entities, such as timed MSC, timed LSCs and UML timed sequence diagrams, are commonly used to express timed scenarios. Common to these approaches is that they usually describe interaction scenarios as black-boxes. Early variants of MSC [MS93, IT96, VGO98] offer very basic support of time, while more recent approaches such as [LMH00, ZK02, KC06] offer a rich set of time construct allowing for more expressiveness. Timed LSC [KW01, HM02] extended MSC capabilities by introducing the concept of liveness and dynamicity. Given these additional concepts, complex timing constraints can be expressed using *hot* and *cold* conditions. On the UML front, the introduction of new time-related data types (e.g. *Time* and *TimeExpression*) in UML2 allowed for attaching timing statements to sequence diagrams. It is worth noting that MSC and UML2 sequence diagrams are the most extensively used scenario notations within industrial software development practices.

Usually deployed during the detailed design as part of the system development life cycle, UML timed statecharts, ROOMCharts, timed BT and timed SDL represent a class of timed scenarios notations that focus on internal behavior of interacting components (i.e. white-box). UML timed statechart approaches [KP91, MSP96] and SDL based approaches [DHHMC95, BGM<sup>+</sup>01, Gra02] are based on a dense time model, supporting a rich set of time constructs as well as the concept of urgency. ROOMcharts [SGW94], based on extended finite state machines (EFSM), do not enforce timing constraints. Instead, ROOM uses MSC concepts to model time constraints [SFR97].

Unlike MSC-based and statechart-based approaches, timed Petri Nets [Ram74, CMS99], timed UCMs [HRD06, Has08], and timed UML Activity Diagrams [Esh02, GM05, LMY<sup>+</sup>01] support component-independent scenarios. This approach allows for the early specification of time-based requirements, without the need to commit already to a specific architecture. Timed UCM and timed activity diagram models offer a fairly rich set of timed constructs, allowing for early stages quantitative analysis of timed systems. Timed Petri Nets have well-established mathematical foundations and offer rich analysis capabilities. At a later stage of the design process, these models can be refined into more detailed design models such as timed MSCs.

The OMG Unified modeling language (UML) considers real-time aspects as part of the profile for Schedulability, Performance and Time (SPT) [OMG02] and more recently it was also included in the profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [OMG07a]. These profiles provide stereotypes to describe concepts such as worst case execution time (WCET) and laxity of deadlines to support quantitative analysis of UML

models (e.g schedulability analysis). Other non OMG UML-based profiles, such as OMEGART [OME07], TURTLE [ACLdSS04] and IF language [OGO06], have proposed several extensions/modifications of timed UML models in order to serve specific needs. The resulting models were mapped to different formalisms (e.g Timed Automata, CTL, etc.) to allow for formal validation and verification. In addition to the model's embedded timed constructs, users may define their own time constraints using OCL [LMM05, Fla03, CK02, FM02b, FM02a].

Component-independent event-based Visual Timed Event Scenarios (VTS) [BKO05] and Real-time Graphical Interval Logic (RTGIL) [MRK<sup>+</sup>97] are designed to express complex properties of real-time systems that allow for model checking. In VTS [BKO05] and RTGIL [MRK<sup>+</sup>97], events are defined and modeled at a more abstract level compared to the MSC-like messages exchanged between components. RTGIL is sufficiently expressive, however, its graphical notation is very similar to temporal logic syntax, making it difficult from a user perspective to comprehend. In contrast, Use Case Maps notation has been successfully used to describe specification patterns [HRD09] in order to improve ease of use.

The Property Sequence Charts (PSC) language [AIP07, ZLS08] positions itself between message based interaction notations and property specification notations. Timed PSC [ZLS08] retain many features from formalisms like MSC, UML sequence diagrams, VTS and time line. Furthermore, PSC models inherit some graphical elements from MSC and UML sequence diagrams, and incorporates message types from time line notation [SHE01] instead of UML *assert*, *optional*, and *neg* frames.

Finally, Regular Timing Diagrams (RTD) [AEKN01] and Action Diagrams (AD) [KC94, KC98], are commonly used notations in the context of hardware design. They describe temporal communication between a system and its environment using changes of signal values at communicating ports.

Even though the surveyed approaches are quite different in terms of their time expressiveness and granularity levels supported, some interesting commonalities can be identified:

- Most of the surveyed construction approaches have time added orthogonally and their untimed syntax can be restored simply by removing the timing constructs.
- In order to allow for quantitative analysis, most timed scenario approaches are mapped into formal models such as timed automata [AD94]. Furthermore, the addition of time gave rise to new variants of verification issues such as timed scenario matching problem [CM06].

## 6. Conclusion

The need to incorporate non-functional aspects, and in particular time-related aspects into requirement languages has been widely recognized. In this paper, we have presented a state of the art survey of forty-seven time-based construction approaches based on fifteen different timed scenario notations (section 2). For the comparison of these construction approaches, we applied the eleven evaluation criteria for modeling time in scenarios introduced in our article (section 3). This state of the art survey and comparison provide users with a much need tool to make an informed decision with respect to selecting a notation that meets their specific needs for modeling time behavior in scenarios.

As part of the future work, it would be interesting to establish meaningful case studies that allow us to evaluate some of our evaluation criteria on real world examples. Furthermore, we

would like to further extend the presented evaluation criteria to include additional criteria such as cost/benefit aspects and usability within the industry.

## References

- [ACldSS04] Ludovic Apvrille, Jean-Pierre Courtiat, Christophe Lohr, and Pierre de Saqui-Sannes. TURTLE: A Real-Time UML Profile Supported by a Formal Validation Toolkit. *IEEE Transactions on Software Engineering*, 30(7):473–487, 2004.
- [AD94] Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
- [AE03] Daniel Amyot and Armin Eberlein. An Evaluation of Scenario Notations and Construction Approaches for Telecommunication Systems Development. *Telecommun. Syst.*, 24(1):61–94, 2003.
- [AEG<sup>+</sup>98] Martin Arnold, M. Erdmann, Martin Glinz, Peter Haumer, Rolf Knoll, Barbara Paech, Klaus Pohl, Johannes Rysler, Rudi Studer, and Klaus Weidenhaupt. Survey on the Scenario Use in Twelve Selected Industrial Projects. Technical Report AIB-07-1998, RWTH Aachen, 1998.
- [AEKN01] Nina Amla, E. Allen Emerson, Robert P. Kurshan, and Kedar S. Namjoshi. Rtdt: A Front-End for Efficient Model Checking of Synchronous Timing Diagrams. In *CAV '01: Proceedings of the 13th International Conference on Computer Aided Verification*, pages 387–390, London, UK, 2001. Springer-Verlag.
- [AEN99] Nina Amla, E. Allen Emerson, and Kedar S. Namjoshi. Efficient Decompositional Model Checking for Regular Timing Diagrams. In *Conference on Correct Hardware Design and Verification Methods*, pages 67–81, 1999.
- [AH97] Rajeev Alur and Thomas A. Henzinger. Real-Time System = discrete system + clock variables. Technical Report UCB/ERL M97/78, EECS Department, University of California, Berkeley, 1997.
- [AHP96] Rajeev Alur, Gerard J. Holzmann, and Doron Peled. An Analyzer for Message Sequence Charts. *Software - Concepts and Tools*, 17(2):70–77, 1996.
- [AIP07] Marco Autili, Paola Inverardi, and Patrizio Pelliccione. Graphical Scenarios for Specifying Temporal Properties: an Automated Approach. *Autom. Softw. Eng.*, 14(3):293–340, 2007.
- [All81] James F. Allen. An interval-based representation of temporal knowledge. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81), Vancouver, BC, Canada, August 1981*, pages 221–226. William Kaufmann, 1981.
- [ALSS<sup>+</sup>06] Apvrille, Ludovic, Saqui-Sannes, Pierre, Khendek, and Ferhat. TURTLE-P: a UML Profile for the Formal Validation of Critical and Distributed Systems. *Software and Systems Modeling (SoSyM)*, 5(4):449–466, December 2006.
- [AMPF07] Charles André, Frédéric Mallet, and Marie-Agnès Peraldi-Frati. Multiform Time in UML for Real-Time Embedded Applications. In *RTCSA: 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007), Daegu, Korea*, pages 232–240, 2007.
- [BAL97] Hanene Ben-Abdallah and Stefan Leue. Expressing and Analyzing Timing Constraints in Message Sequence Chart Specifications. Technical Report 97-04, Department of Electrical and Computer Engineering, University of Waterloo, 1997.
- [Ber03] Kirsten Berkenkotter. Using UML 2.0 in Real-Time Development: A Critical Review. In *Workshop Specification and Validation of UML models for Real Time and Embedded Systems (SVERTS'03) at UML'03*, pages 41–54, San Francisco, CA, USA, October 2003.
- [BFG<sup>+</sup>99] Marius Bozga, Jean-Claude Fernandez, Lucian Ghirvu, Susanne Graf, Jean-Pierre Krimm, and Laurent Mounier. IF: An Intermediate Representation and Validation Environment for Timed Asynchronous Systems. In *World Congress on Formal Methods (1)*, pages 307–327, 1999.
- [BFG<sup>+</sup>00] Marius Bozga, Jean-Claude Fernandez, Lucian Ghirvu, Susanne Graf, Jean-Pierre Krimm, and Laurent Mounier. IF: A Validation Environment for Timed Asynchronous Systems. In *Computer Aided Verification*, pages 543–547, 2000.
- [BG06] Howard Bowman and Rodolfo Gomez. *Concurrency Theory - Calculi and Automata for Modelling Untimed and Timed Concurrent Systems*. Springer-Verlag, 2006.
- [BGM<sup>+</sup>01] Marius Bozga, Susanne Graf, Laurent Mounier, Iulian Ober, Jean-Luc Roux, and Daniel Vincent. Timed Extensions for SDL. In *SDL '01: Proceedings of the 10th International SDL Forum Copenhagen on Meeting UML*, pages 223–240, London, UK, 2001. Springer-Verlag.
- [BK94] Jixin Brian Knight. Time Representation: A Taxonomy of Temporal Models. *AI Review*, 7:401–419, 1994.
- [BKO05] Victor Braberman, Nicolas Kicillof, and Alfredo Olivero. A Scenario-Matching Approach to the Description and Model Checking of Real-Time Properties. *IEEE Transactions on Software Engineering*, 31(12):1028–1041, 2005.
- [BL92] Tommaso Bolognesi and Ferdinando Lucidi. LOTOS-like Process Algebras with Urgent or Timed Interactions. In *FORTE '91: Proceedings of the IFIP TC6/WG6.1 Fourth International Conference on Formal*

- Description Techniques for Distributed Systems and Communication Protocols*, pages 249–264, Amsterdam, The Netherlands, The Netherlands, 1992. North-Holland Publishing Co.
- [BM07] Simona Bernardi and José Merseguer. A UML profile for dependability analysis of real-time embedded systems. In *WOSP '07: Proceedings of the 6th international workshop on Software and performance*, pages 115–124, New York, NY, USA, 2007. ACM Press.
- [BRJ96] Grady Booch, James Rumbaugh, and Ivar Jacobson. Unified Modeling Language for Object-Oriented Development (version 0.91 addendum), 1996.
- [BS00] Sébastien Bornot and Joseph Sifakis. An Algebraic Framework for Urgency. *Inf. Comput.*, 163(1):172–202, 2000.
- [BS04] Dorina Petriu Bernardi Simona. Comparing Two UML Profiles For Non-functional Requirement Annotations: the SPT and QoS Profiles. In *SVERTS - Satellite Events at the UML Conference*, Portugal, 2004.
- [BST98] Sébastien Bornot, Joseph Sifakis, and Stavros Tripakis. Modeling Urgency in Timed Systems. *Lecture Notes in Computer Science*, 1536:103–129, 1998.
- [CFP01] Flavio Corradini, Gian Luigi Ferrari, and Marco Pistore. On the Semantics of Durational Actions. *Theoretical Computer Science*, 269(1-2):47–82, 2001.
- [CGW08] Robert Colvin, Lars Grunske, and Kirsten Winter. Timed Behavior Trees for Failure Mode and Effects Analysis of Time-Critical Systems. *Journal of Systems and Software*, 81(12):2163–2182, 2008.
- [CHR91] Zhou Chaochen, C. A. R. Hoare, and Anders P. Ravn. A Calculus of Durations. *Inf. Process. Lett.*, 40(5):269–276, 1991.
- [CK02] María Victoria Cengarle and Alexander Knapp. Towards OCL/RT. In *FME '02: Proceedings of the International Symposium of Formal Methods Europe on Formal Methods - Getting IT Right*, pages 390–409, London, UK, 2002. Springer-Verlag.
- [CM99] Brian D. Chance and Bonnie E. Melhart. A Taxonomy for Scenario Use in Requirements Elicitation and Analysis of Software Systems. *6th Symposium on Engineering of Computer-Based Systems (ECBS'99), 7-12 March 1999, Nashville, TN, USA. IEEE Computer*, 00:232, 1999.
- [CM06] Prakash Chandrasekaran and Madhavan Mukund. Matching Scenarios with Timing Constraints. In *Formal Modeling and Analysis of Timed Systems*, volume 4202 of *Lecture Notes in Computer Science*, pages 98–112. Springer, 2006.
- [CMS99] Antonio Cerone and Andrea Maggiolo-Schettini. Time-based Expressivity of Timed Petri Nets for System Specification. *Theor. Comput. Sci.*, 216(1-2):1–53, 1999.
- [Coc97] Alistair Cockburn. Structuring Use Cases with Goals. *Journal of Object-Oriented Programming*, Sept-Oct and Nov-Dec, 1997.
- [Cor00] Flavio Corradini. Absolute versus Relative Time in Process Algebras. *Inf. Comput.*, 156(1-2):122–172, 2000.
- [CP04] Vittorio Cortellessa and Antonio Pompei. Towards a UML profile for QoS: a contribution in the reliability domain. In *WOSP '04: Proceedings of the 4th international workshop on Software and performance*, pages 197–206, New York, NY, USA, 2004. ACM Press.
- [CRH98] Corine Cauvet Jolita Ralyté Alistair G. Sutcliffe Neil A. M. Maiden Matthias Jarke Peter Haumer Klaus Pohl Eric Dubois Colette Rolland, Camille B. Achour and Patrick Heymans. A Proposal for a Scenario Classification Framework. *Requirements Engineering*, 3(1):23–47, 1998.
- [DHHMC95] Marc Diefenbruch, Elke Heck, Jörg Hintelmann, and Bruno Müller-Clostermann. Performance evaluation of SDL systems adjunct by queuing models. In *Proc. of SDL-Forum '95*, London, UK, 1995. Springer-Verlag.
- [Die96] Cheryl Dietz. Graphical formalization of real-time requirements. In *FTRTFT '96: Proceedings of the 4th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 366–384, London, UK, 1996. Springer-Verlag.
- [Dro03] R. Geoff Dromey. From requirements to design: Formalizing the key steps. In *1st International Conference on Software Engineering and Formal Methods (SEFM 2003)*, pages 2–, 2003.
- [EDG<sup>+</sup>05] Huáscar Espinoza, Hubert Dubois, Sébastien Gérard, Julio L. Medina Pasaje, Dorina C. Petriu, and C. Murray Woodside. Annotating UML models with non-functional properties for quantitative analysis. In *MoD-ELS Satellite Events*, pages 79–90, 2005.
- [EE04] EAST-EEA. EAST-EEA. embedded electronic architecture. <http://www.east-eea.net>, 2004.
- [EERdR01] W. J. Toetenel Ella E. Roubtsova, Jan van Katwijk and Ruud C. M. de Rooij. Real-Time Systems: Specification of Properties in UML. In *ASCI 2001 conference, Het Heijderbos, Heijen, The Netherlands, May 30 - June 1 2001*, pages 188–195, 2001.
- [EMCGP99] Jr. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.
- [Esh02] Rik Eshuis. *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*. PhD thesis, University of Twente, Enschede, The Netherlands, 2002.

- [EW01] Rik Eshuis and Roel Wieringa. A formal semantics for UML activity diagrams – formalising workflow models. Technical report, University of Twente, Department of Computer Science, University of Twente, 2001.
- [FHD<sup>+</sup>99] Thomas Firley, Michaela Huhn, Karsten Diethers, Thomas Gehrke, and Ursula Goltz. Timed sequence diagrams and tool-based analysis A case study. In Robert France and Bernhard Rumpe, editors, *UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference, Fort Collins, CO, USA, October 28-30, 1999, Proceedings*, volume 1723, pages 645–660. Springer, 1999.
- [Fla03] Stephan Flake. Temporal OCL extensions for specification of real-time constraints. In *Workshop Specification and Validation of UML models for Real Time and Embedded Systems (SVERTS'03) at UML'03*, San Francisco, CA, USA, October 2003.
- [FM02a] Stephan Flake and Wolfgang Mueller. A UML Profile for Real-Time Constraints with the OCL. In *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*, pages 179–195, London, UK, 2002. Springer-Verlag.
- [FM02b] Stephan Flake and Wolfgang Mueller. Specification of Real-Time Properties for UML Models. In *HICSS '02: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)- Volume 9*, page 277, Washington, DC, USA, 2002. IEEE Computer Society.
- [GK06] Abdelouahed Gherbi and Ferhat Khendek. UML profiles for real-time systems and their applications. *Journal of Object Technology*, 5(4):149–169, 2006.
- [GL99] Mark K. Gardner and Jane W.-S. Liu. Analyzing Stochastic Fixed-Priority Real-Time Systems. In *TACAS '99: Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pages 44–58, London, UK, 1999. Springer-Verlag.
- [GM05] Nicolas Guelfi and Amel Mammar. A Formal Semantics of Timed Activity Diagrams and its PROMELA Translation. In *APSEC'05: Proceedings of the 12th Asia-Pacific Software Engineering Conference*, pages 283–290, Washington, DC, USA, 2005. IEEE Computer Society.
- [GOO06] Susanne Graf, Ileana Ober, and Iulian Ober. A Real-time Profile for UML. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(2):113–127, 2006.
- [Gra02] Susanne Graf. Expression of Time and Duration Constraints in SDL. In *3rd SAM Workshop on SDL and MSC, University of Wales Aberystwyth*, number 2599 in LNCS, June 2002.
- [GRS95] Roberto Gorrieri, Marco Roccetti, and Enrico Stancampiano. A Theory of Processes with Durational Actions. *Theoretical Computer Science*, 140(1):73–94, 1995.
- [Gur88] Yuri Gurevich. Logic and the Challenge of Computer Science. In E. Börger, editor, *Current trends in theoretical computer science*, pages 1–57. Computer Science Press, 1988.
- [GWC07] Lars Grunske, Kirsten Winter, and Robert Colvin. Timed Behavior Trees and Their Application to Verifying Real-Time Systems. In *ASWEC '07: Proceedings of the 2007 Australian Software Engineering Conference*, pages 211–222, Washington, DC, USA, 2007. IEEE Computer Society.
- [Har87] David Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, June 1987.
- [Has08] Jameleddine Hassine. *Formal Semantics and Verification of Use Case Maps*. PhD thesis, Concordia University, Montreal, Quebec, Canada, 2008.
- [HHRS05] Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. STAIRS towards Formal Design with Sequence Diagrams. *Software and System Modeling*, 4(4):355–357, 2005.
- [HM01] Davide Harel and Rami Marelly. Specifying and Executing Behavioral Requirements: The Play-In/Play-Out Approach. Technical Report MSC01-15, The Weizmann Institute of Science, 2001.
- [HM02] David Harel and Rami Marelly. Playing with Time: On the Specification and Execution of Time-Enriched LSCs. In *MASCOTS '02: Proceedings of the 10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'02)*, page 193, Washington, DC, USA, 2002. IEEE Computer Society.
- [Hoa85] Charles Antony Richard Hoare. *Communicating Sequential Processes*. Prentice/Hall International, April 1985.
- [HR95] Matthew Hennessy and Tim Regan. A Process Algebra for Timed Systems. *Inf. Comput.*, 117(2):221–239, 1995.
- [HR00] David Harel and Bernhard Rumpe. Modeling languages: Syntax, semantics and all that stuff, part i: The basic stuff. Technical Report MCS00-16, Mathematics & Computer Science, Weizmann Institute Of Science, Jerusalem, Israel, Israel, 2000.
- [HRD05a] Jameleddine Hassine, Juergen Rilling, and Rachida Dssouli. Abstract Operational Semantics for Use Case Maps. In *Formal Techniques for Networked and Distributed Systems - FORTE 2005, 25th IFIP WG 6.1 International Conference, Taipei, Taiwan, October 2-5*, pages 366–380, 2005.
- [HRD05b] Jameleddine Hassine, Juergen Rilling, and Rachida Dssouli. An ASM Operational Semantics for Use Case Maps. In *RE '05: Proceedings of the 13th IEEE International Conference on Requirements Engineering*

- (RE'05), Paris, pages 467–468. IEEE Computer Society, 2005.
- [HRD06] Jameleddine Hassine, Juergen Rilling, and Rachida Dssouli. Timed Use Case Maps. In *System Analysis and Modeling: Language Profiles, 5th International Workshop, SAM 2006, Kaiserslautern, Germany, May 31 - June 2, 2006, Revised Selected Papers*, pages 99–114, 2006.
- [HRD07] Jameleddine Hassine, Juergen Rilling, and Rachida Dssouli. Formal Verification of Use Case Maps with Real Time Extensions. In *SDL 2007: Design for Dependable Systems, 13th International SDL Forum, Paris, France, September 18-21, 2007, Proceedings*, pages 225–241, 2007.
- [HRD09] Jameleddine Hassine, Juergen Rilling, and Rachida Dssouli. Use Case Maps as a Property Specification Language. *Software and System Modeling*, 8(2):205–220, April 2009.
- [ISO97] ISO. Time extended LOTOS. international standards organization, 1997.
- [IT96] ITU-T. Recommendation Z.120. Message Sequence Charts (MSC). Geneva, Switzerland, 1996.
- [IT02] ITU-T. Specification and Description Language, Recommendation Z.100 (SDL). Geneva, Switzerland, 2002.
- [IT04] ITU-T. Recommendation Z.120 (04/04). Message Sequence Charts (MSC). Geneva, Switzerland, 2004.
- [IT08] ITU-T. Draft recommendation Z.151, User Requirements Notation (URN), 2008.
- [ITE04] ITEA. ITEA: EAST-ADL – the EAST-EEA Architecture Description Language. ITEA Project Version 1.02, 2004.
- [JBR99] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
- [JEJ94] Ivar Jacobson, Maria Ericsson, and Agneta Jacobson. *The Object Advantage: Business Process Reengineering with Object Technology*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1994.
- [JP86] Mathai Joseph and Paritosh K. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, 29(5):390–395, 1986.
- [KC94] Karim Khordoc and Eduard Cerny. Modeling cell processing hardware with action diagrams. In *ISCAS*, pages 245–248, 1994.
- [KC98] Karim Khordoc and Eduard Cerny. Semantics and Verification of Action Diagrams with Linear Timing. *ACM Trans. Des. Autom. Electron. Syst.*, 3(1):21–50, 1998.
- [KC06] Tai Hyo Kim and Sung Deok Cha. Timed High-Level Message Sequence Charts for Real-Time System Design. In *System Analysis and Modeling: Language Profiles, 5th International Workshop, SAM 2006, Kaiserslautern, Germany, May 31 - June 2, 2006, Revised Selected Papers*, pages 82–98, 2006.
- [KCH01] Saehwa Kim, Sukjae Cho, and Seongsoo Hong. Automatic Implementation of Real-Time Object-Oriented Models and Schedulability Issues. In *WORDS '01: Proceedings of the Sixth International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'01)*, page 137, Washington, DC, USA, 2001. IEEE Computer Society.
- [KdB04] Marcel Kyas and Frank S. de Boer. On Message Specifications in OCL. *Electr. Notes Theor. Comput. Sci.*, 101:73–93, 2004.
- [Kho96] Karim Khordoc. *Action Diagrams: A Methodology for the Specification and Verification of Real-time Systems*. PhD thesis, McGill University, Montreal, Canada, 1996.
- [KP91] Yonit Kesten and Amir Pnueli. Timed and Hybrid Statecharts and Their Textual Representation. In *Proceedings of the Second International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 591–620, London, UK, 1991. Springer-Verlag.
- [KW01] Jochen Klose and Hartmut Wittke. An Automata Based Interpretation of Live Sequence Charts. In *TACAS 2001: Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 512–527, London, UK, 2001. Springer-Verlag.
- [Lam78] Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM*, 21(7):558–565, 1978.
- [LDD06] Hongzhi Liang, Juergen Dingel, and Zinovy Diskin. A Comparative Survey of Scenario-Based to State-Based Model Synthesis Approaches. In *SCESM '06: Proceedings of the 2006 international workshop on Scenarios and state machines: models, algorithms, and tools*, pages 5–12. ACM Press, 2006.
- [LL73] C. L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.
- [LL94] Luc Léonard and Guy Leduc. An Enhanced Version of Timed LOTOS and its Application to a Case Study. In *FORTE '93: Proceedings of the IFIP TC6/WG6.1 Sixth International Conference on Formal Description Techniques, VI*, pages 483–498, Amsterdam, The Netherlands, The Netherlands, 1994. North-Holland Publishing Co.
- [LL99a] Xuandong Li and Johan Lilius. Timing Analysis of Message Sequence Charts. Technical Report TUCS-TR-255, TUCS - Turku Centre for Computer Science, 24, 1999.
- [LL99b] Xuandong Li and Johan Lilius. Timing analysis of UML sequence diagrams. In Robert France and Bernhard Rumpe, editors, *UML'99 - The Unified Modeling Language. Beyond the Standard. Second International*

- Conference, Fort Collins, CO, USA, volume 1723, pages 661–674. Springer, 1999.
- [LMH00] P. Le Maigat and L. Hérouët. A (max,+) Approach for Time in Message Sequence Charts. In R. Boel and G. Stremersch, editors, *Proceedings of the 5th Workshop on Discrete Event Systems*, pages 83–92, Ghent, Belgium, 2000. Kluwer Academic Publishers.
- [LMM05] Luigi Lavazza, Sandro Morasca, and Angelo Morzenti. A Dual Language Approach to the Development of Time-Critical Systems. *Electr. Notes Theor. Comput. Sci.*, 116:227–239, 2005.
- [LMY+01] Xuandong Li, Cui Meng, Pei Yu, Jianhua Zhao, and Guoliang Zheng. Timing analysis of UML activity diagrams. In *UML 2001 - The Unified Modeling Language, Modeling Languages, Concepts, and Tools, 4th International Conference, Toronto, Canada, October 1-5, 2001, Proceedings*, pages 62–75, London, UK, 2001. Springer-Verlag.
- [LPY97] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.
- [Mai98] N. A. M. Maiden. Crews-savre: Scenarios for Acquiring and Validating Requirements. *Automated Software Engineering*, 5(4):419–446, 1998.
- [Mer74] Philip Meir Merlin. *A Study of the Recoverability of Computing Systems*. PhD thesis, University of California, Irvine, 1974.
- [Mes90] David G. Messerschmitt. Synchronization in Digital System Design. *IEEE Journal on Selected Areas in Communications*, 8(8):1404–1419, 1990.
- [MGHL94] Mark H. Klein Michael Gonzalez Harbour and John P. Lehoczky. Timing Analysis for Fixed-Priority Scheduling of Hard Real-Time Systems. *IEEE Trans. Softw. Eng.*, 20(1):13–28, 1994.
- [MLLG01] Grant Martin, Luciano Lavagno, and Jean Louis-Guerin. Embedded UML: a Merger of Real-time UML and co-design. In *CODES '01: Proceedings of the ninth international symposium on Hardware/software codesign*, pages 23–28, New York, NY, USA, 2001. ACM Press.
- [MP96] Zohar Manna and Amir Pnueli. Clocked Transition Systems. Technical report, Stanford University, Stanford, CA, USA, 1996.
- [MRK+97] Louise E. Moser, Y. S. Ramakrishna, G. Kutty, P. M. Melliar-Smith, and Laura K. Dillon. A Graphical Environment for the Design of Concurrent Real-time Systems. *ACM Transactions on Software Engineering and Methodology*, 6(1):31–79, 1997.
- [MS93] N. Meng-Siew. Reasoning with Timing Constraints in Message Sequence Charts. Master’s thesis, University of Stirling, Scotland, U.K., August 1993.
- [MSP96] Andrea Maggiolo-Schettini and Adriano Peron. Retiming Techniques for Statecharts. In *FTRTFT '96: Proceedings of the 4th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 55–71, London, UK, 1996. Springer-Verlag.
- [Nat85] Stéphane Natkin. Timed and Stochastic Petri Nets: From the Validation to the Performance of Synchronization Schemes. In *International Workshop on Timed Petri Nets*, pages 2–3. IEEE Computer Society, 1985.
- [NS92] Xavier Nicollin and Joseph Sifakis. An Overview and Synthesis on Timed Process Algebras. In *Proceedings of the Real-Time: Theory in Practice, REX Workshop*, pages 526–548, London, UK, 1992. Springer-Verlag.
- [NS94] Xavier Nicollin and Joseph Sifakis. The Algebra of Timed Processes, ATP: Theory and Application. *Inf. Comput.*, 114(1):131–178, 1994.
- [OGO06] Iulian Ober, Susanne Graf, and Ileana Ober. Validating timed UML Models by Simulation and Verification. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(2):128–145, 2006.
- [OME07] OMEGA. OMEGA consortium. webpage of the omega ist project. <http://www-omega.imag.fr/>, 2007. Last accessed, March 2009.
- [OMG02] OMG. Response to the OMG RFP for Schedulability, Performance and Time, v. 2.0. OMG document ad/2002-03-04, March 2002.
- [OMG03] OMG. Unified Modeling Language specification, 2003. Version 1.5, March 2003 via <http://www.omg.org>.
- [OMG06] OMG. Object management group. UML profile for modeling quality of service and fault tolerant characteristics and mechanisms. OMG document formal/06-05-02, May 2006.
- [OMG07a] OMG. MARTE OMG Specification. A UML Profile for MARTE. Beta 1. OMG Adopted specification ptc/07-08-04, August 2007.
- [OMG07b] OMG Management Group. UML 2.2.1 Superstructure Specification, oct 2007.
- [PB00] Peter Puschner and Alan Burns. A Review of Worst-Case Execution-Time Analysis. *Journal of Real-Time Systems*, 18(2/3):115–128, May 2000.
- [PFS08] M.-A. Peraldi-Frati and Y. Sorel. From High-Level Modelling of Time in MARTE to Real-Time Scheduling Analysis. In *Proceedings of MODELS'08 Workshop on Architecting and Construction of Embedded Systems-Model Based, ACES-MB'08*, Toulouse, France, SEP 2008.
- [PIM08] Patrizio Pelliccione, Paola Inverardi, and Henry Muccini. CHARMY: A Framework for Designing and

- Verifying Architectural Specifications. *IEEE Transactions on Software Engineering*, 99(2), 2008.
- [PMS94] Adriano Peron and Andrea Maggiolo-Schettini. Transitions as Interrupts: A New Semantics for Timed Statecharts. In *TACS '94: Proceedings of the International Conference on Theoretical Aspects of Computer Software*, pages 806–821, London, UK, 1994. Springer-Verlag.
- [Ram74] Chandar Ramchandani. Analysis of Asynchronous Concurrent Systems by Timed Petri Nets. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1974.
- [RJB99] James Rumbaugh, Ivar Jacobson, and Grady Booch, editors. *The Unified Modeling Language Reference Manual*. Addison-Wesley Longman Ltd., Essex, UK, 1999.
- [RL07a] RT-LOTOS. CADP. Construction and Analysis of Distributed Processes. <http://www.inrialpes.fr/vasy/cadp/>, 2007. Last accessed, March 2009.
- [RL07b] RT-LOTOS. RT-LOTOS. software and tools for communicating systems. <http://www.laas.fr/RT-LOTOS/>, 2007. Last accessed, March 2009.
- [RMSM<sup>+</sup>96] Y. S. Ramakrishna, P. M. Melliar-Smith, Louise E. Moser, Laura K. Dillon, and G. Kuty. Interval Logics and Their Decision Procedures, Part I: An Interval Logic. *Theor. Comput. Sci.*, 166(1&2):1–47, 1996.
- [RP96] Colette Rolland and Naveen Prakash. A Proposal for Context-Specific Method Engineering. In *Proceedings of the IFIP TC8, WG8.1/8.2 working conference on method engineering on Method engineering : principles of method construction and tool support*, pages 191–208, London, UK, UK, 1996. Chapman & Hall, Ltd.
- [RR88] George M. Reed and A. W. Roscoe. A Timed Model for Communicating Sequential Processes. *Theoretical Computer Science*, 58(1-3):249–261, 1988.
- [RR98] Björn Regnell and Per Runeson. Combining Scenario-based Requirements with Static Verification and Dynamic Testing. In *Proc. Fourth Intern. Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ98)*, 1998.
- [SDV95] Stephane Somé, Rachida Dssouli, and Jean Vaucher. From Scenarios to Timed Automata: Building Specifications from Users Requirements. In *APSEC '95: Proceedings of the Second Asia Pacific Software Engineering Conference*, page 48, Washington, DC, USA, 1995. IEEE Computer Society.
- [SDV96] Stephane Somé, Rachida Dssouli, and Jean Vaucher. Toward an Automation of Requirements Engineering using Scenarios. *Journal of Computing and Information*, 2(1):1110–1132, 1996.
- [SFR97] Manas Saksena, Paul Freedman, and Pawel Rodziewicz. Guidelines for Automated Implementation of Executable Object Oriented Models for Real-Time Embedded Control Systems. In *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS97)*, page 240. IEEE Computer Society, 1997.
- [SGW94] Bran Selic, Garth Gullekson, and Paul T. Ward. *Real-Time Object-Oriented Modeling*. John Wiley & Sons, Inc., New York, NY, USA, 1994.
- [SHE01] Margaret H. Smith, Gerard J. Holzmann, and Kousha Eteessami. Events and Constraints: A Graphical Editor for Capturing Logic Requirements of Programs. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, pages 14–22, Washington, DC, USA, 2001. IEEE Computer Society.
- [Sin04] Richard O. Sinnott. The Formal, Tool Supported Development of Real Time Systems. In *SEFM '04: Proceedings of the Software Engineering and Formal Methods, Second International Conference on (SEFM'04)*, pages 388–395, Washington, DC, USA, 2004. IEEE Computer Society.
- [SKW00] Manas Saksena, Panagiota Karvelas, and Yun Wang. Automatic Synthesis of Multi-Tasking Implementations from Real-Time Object-Oriented Models. In *ISORC '00: Proceedings of the Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, page 360, Washington, DC, USA, 2000. IEEE Computer Society.
- [Som04] Stephane Somé. An environment for use cases based requirements engineering. In *RE '04: Proceedings of the Requirements Engineering Conference, 12th IEEE International (RE'04)*, pages 364–365, Washington, DC, USA, 2004. IEEE Computer Society.
- [SS01] Shane Sendall and Alfred Strohmeier. Specifying Concurrent System Behavior and Timing Constraints Using OCL and UML. In *UML 2001 - The Unified Modeling Language, Modeling Languages, Concepts, and Tools, 4th International Conference, Toronto, Canada, October 1-5, 2001, Proceedings*, pages 391–405, London, UK, 2001. Springer-Verlag.
- [TBW94] Ken W. Tindell, Alan Burns, and Andy J. Wellings. An Extendible Approach for Analyzing Fixed Priority Hard Real-time Tasks. *Real-Time Syst.*, 6(2):133–151, 1994.
- [TTo07] TTool. TTool. A Toolkit for Editing and Validating TURTLE Diagrams. <http://labsoc.comelec.enst.fr/turtle/>, 2007. Last accessed, March 2009.
- [VGO98] Cheryl Dietz Volker Grabowski and Ernst-Rudiger Olderog. Semantics for Timed Message Sequence Charts via Constraint Diagrams. In Y. Lahav, A. Wolisz, J. Fischer, and E. Holz, editors, *Proceedings of the 1st Workshop of the SDL Forum Society on SDL and MSC*, pages 251–260. Humboldt-Universitaet zu Berlin/Germany, 1998.
- [WP04] Murray Woodside and Dorina Petriu. Capabilities of the UML Profile for Schedulability Performance and Time (SPT). In *Workshop SIVOES-SPT held in conjunction with the 10th IEEE RTAS 2004*, 2004.

- [ZK02] Tong Zheng and Ferhat Khendek. An Extension for MSC-2000 and Its Application. In *Telecommunications and beyond: The Broader Applicability of SDL and MSC, Third International Workshop, SAM 2002, Aberystwyth, UK, June 24-26. Revised Papers*, pages 221–232, 2002.
- [ZKH02] Tong Zheng, Ferhat Khendek, and Loc H elou et. A Semantics for Timed MSC. *Electr. Notes Theor. Comput. Sci.*, 65(7), 2002.
- [ZLS08] Pengcheng Zhang, Bixin Li, and Mingjie Sun. A Timed Extension of Property Sequence Chart. *High-Assurance Systems Engineering, IEEE International Symposium on*, 0:197–206, 2008.