Expression of time and duration constraints in SDL

Susanne Graf

Verimag, Grenoble, France
Susanne.graf@imag.fr
http://www-verimag.imag.fr/~graf

Abstract. in this paper we give an overview on a set of time related features, useful in the context of real-time system design and classify them into two categories, those needed for modeling of non functional aspects and analysis, and those needed for functional design. We are careful to allow a clear distinction between functional and non functional parts of a specification. We show how these features are represented at the semantic level with a minimal number of primitives.

1. Introduction

The ITU Specification and description language SDL is increasingly used outside its historical domain for the development of real-time and embedded systems, in particular those, where the functional behavior is time dependent, and therefore time plays both a functional and non-functional role which must be clearly distinguished.

SDL is a modeling language in which sufficient details can be given for the generation of code preserving all the properties of the specification, including those concerning timing. For the functional design of a real-time system, sufficient *functional time related primitives* are needed. SDL has already some important time related features, such as a notion of *global time* (allowing to measure durations by means of appropriate time stamps which can be passed throughout the whole system), and the possibility to allow time dependent triggering of transitions (*timeouts* allow to define an earliest triggering time and *enabling conditions* can define general triggering constraints). Explicit means to quit some time consuming activity and to describe systems where time is (partly) under the control of the systems are needed

For modeling nonfunctional aspects for analysis, almost nothing exists in standard SDL. Several proposals exist, which enhance SDL to make time and performance analysis possible. Previous work is mainly dealing with performance evaluation [BB93, SPI97, Rou98, MIT99, Mal99] or on requirements expression [Leu95, ALH95, DDH+01], but there exists also work on timed verification [OCK00], schedulability [ALV99, ADL+99, ADL+01]. A general real time framework for SDL is presented in [SDM+00] and [BGK00, BGM*01], where the first one is meant mainly for hardware software co-design.

Most of these approaches, in particular, the ones mainly concerned by performance analysis advocate scenario based timing information, by means of scenario based languages (such as Message sequence charts or activity diagrams) which provide

timing information for a set of "relevant" scenarios, whereas for all other scenarios (supposed to occur rarely) no timing information is available. Some tools, for example time enhanced versions of ObjectGeode [Geode] as presented in [Rou98, O01] and Tau [Tau] and the tool and methods based on Queuing SDL [QSDL, MIT99] propose to attach explicit timing information with SDL constructs such as tasks, and to provide some minimal deployment information.

Our approach [BGK00,BGM01,D21GEN02] is an extension of these latter approaches, where we are focusing on timing rather than on performance analysis[1]. In section 2, we discuss a minimal set of primitives and how language level real-time concepts can be expressed by means of the semantic level primitives. Concerning the needs for real-time primitives at SDL level, as introduced in [D21GEN02], we clearly distinguish between needs for modeling non functional aspects – discussed in section 3 - and needs for functional design – discussed in section 4, where we motivate the need of concepts, propose a solution and provide an informal mapping to the semantic level.

2. Time in semantic models

At the semantic level, it is interesting to have a minimal number of basic primitives allowing to express all high level time concepts, functional and non functional ones. In fact semantic level models make no distinction between these aspects and in semantic models, time is the object of modeling and can be constraint in various ways.

An interesting semantic framework are timed automata [HNSY92, AH94, BST98, AGS00, BGS00], where

- time progress and system progress are along orthogonal dimensions, such that system transitions are timeless (instantaneous events) and time progresses in system states
- where the system can restrict time progress in states (by means of a notion of urgency)
- and system transitions can be enabled or disabled by time progress.

Timed automata have a notion of global state and an explicit notion of concurrency and the possibility to synchronize transitions in concurrent entities.

The standard semantic of SDL as given in [Z100] is expressed by means of Abstract State Machines [Gur97, EGG*00], which is in fact both a semantic level formalism and modeling language itself. ASM has no predefined time concept attached with it, but is expressive enough to express almost any time semantics. The ASM semantics for SDL as presented in Z100

1. defines the level of atomicity of SDL by cutting each SDL transition into a number of atomic steps, that is a discrete transition between states, where all intermediate auxiliary steps, if they exist, do not appear in the model (the exact granularity is almost purely functional and not the object of the discussion)

[1] Notice that the approach can be adapted to performance analysis in an almost straightforward way by using constraints of probabilistic nature

2. it does not enforce orthogonality of time and system progress, but allows external time progress arbitrarily in both states and transitions.

For any ASM model with time progress in transitions, can be defined an equivalent ASM model with the additional constraint that time does not progress in transitions. Every *atomic step* is split into two *instantaneous atomic step*, and time passes in states only (see the figure below, where possibility of time progress is suggested by time intervals).

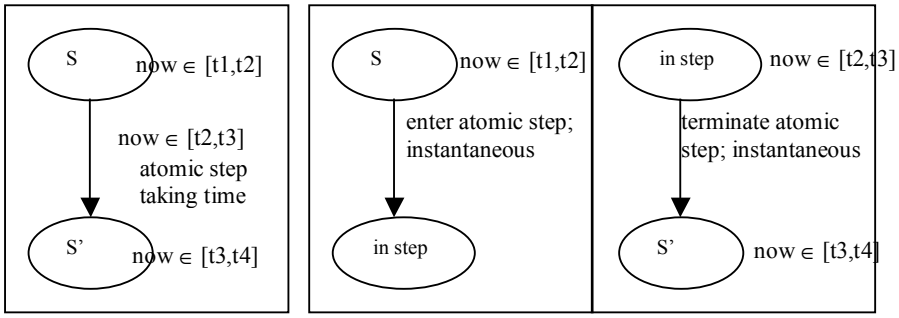


Fig. 1. transforming an atomic step taking time into two instantaneous transitions with time progress in states only

(Communicating) timed automata with urgency [BST98, AGS00, BGS00] are a good abstract model for time, and expressible in ASM. In the remainder of this section we nevertheless prefer to use the timed automata framework to show how time related concepts can be expressed at the semantic level, as they are more intuitive.

Timed automata and modeling languages based upon them [BFG*99,BGM01, BLL*98] provide for measuring time a primitive called "*clock*" instead of providing a global time ***now***. Such a *clock* can be **set** (to zero) to start measuring a duration, consulted for its current **value** (that is the duration since it has been set). The aim of the use of *clocks* is the encapsulation of time stamping for measuring durations without using ***now***. In order not to confuse the reader acquainted with SDL, we use in all examples ***now*** explicitly instead.

At semantic level, a transition (an instantaneous state change of the system, as in Fig.°1) is fully characterized by:

- its functional triggering condition and transition function or relation, which we abstract away for now
- a time (and possibly data) dependent enabling condition, expressing at which time the transition is possible
- an urgency attribute which is either *lazy, delayable* or *eager* where
    - *lazy* transitions can wait forever. Whenever a lazy transition is enabled, it can be taken, or likewise time can progress and possibly disable it. This is the default whenever time is considered as external to the system
    - *eager* transitions never wait. When an eager transition is enabled, only instantaneous discrete transitions are possible as long as any *eager* transitions are enabled.

- *delayable* transitions can wait, but only until the falling edge of their enabling condition, that is they can never wait until disabled by time progress.

Urgency allows to control time progress at the semantic level in a very general, flexible and compositional way [BST98,AGS00]. Formally, the urgency of all outgoing transitions of a (global) system state, impose the following restriction on possible time progress in **s**: at time point **t**, waiting is possible for a duration **δ**, as long as the Time Progress Condition

$$\mathbf{TPC}(\mathbf{s})(\mathbf{t})(d) = \bigwedge_{tr \in trans(s)} \neg enabled(tr)(\mathbf{t}+d) \vee \neg urgent(tr)(\mathbf{t}+d)$$

holds for every duration $\mathbf{d<\delta}$. The predicate urgent(tr) expresses when a transition id *urgent*, that is equivalent to false for *lazy* transition, equivalent to the enabledness predicate for *eager* transition, and expresses the "falling edge" of the enabledness predicate for *delayable* transitions (which is not allowed to be strict).

An atomic ASM step which is executed before time t_0, and which has a duration of 2 to 3 time units, is modeled as in Fig. 2: the control over the *duration* of the step is expressed exactly in the same way as the control over the *starting time* of the step, namely by a *delayable* time constraint on the next transition; that is the end of the duration of the atomic step is defined by the point of time at which the finish transition is triggered. A

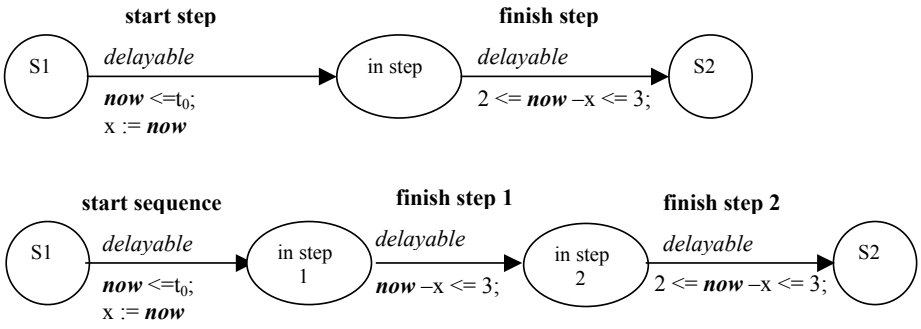


Fig.2: representing an atomic step with time constraints

The duration of a sequence of steps can be constrained in a similar way as shown in Fig.2[2].: the intermediate steps can occur at any time, but not later than the maximal overall duration, and the last step must additionally satisfy the required minimal duration of the overall sequence of steps. This does not exclude the sequence where the duration has reached its maximal value already in the starting state *s1*; this means just that time will not progress any further until the end of the sequence. This is

[2] Where, in order to keep the figure reasonably small, the internal states "*before entering step i*" and "*in step I*" are grouped into a single state, which is a correct optimization when the starting transitions of intermediate steps are *eager*, or when they are *unobservable*; this because waiting for duration d_1 followed by waiting for duration d_2 is equivalent to waiting for duration d_1 and d_2.

especially interesting when the intermediate states are considered as "*unobservable*" (for verification) and can altogether be identified with state *s1*.

Also *duration constraints on communications* are expressed in the very same way at the semantic level, where a channel with non zero delay can modeled

- As in timed automata, by an explicit automaton, receiving signals at the time they are sent, and inserting them into the input queue of the receiver process when the time constraint on the communication delay is satisfied
- or as in the ASM SDL semantics, by a sequence of instantaneous atomic steps triggered by the output command, which insert the signal into the input queue of the receiver and associate with it an arrival time in the future.

The only potential problem of the second model appears when the receiver does not yet exist at sending time, but does exist at reception time.

Both semantics allow to represent variants of channels, allowing losses, reordering,…in a straightforward manner.

From a modeling point of view it is interesting to distinguish channels in which the communication delay of each signal is independent of other signals in transit (that is all time guards are independent of each other, except that their order is preserved), and those in which there exists some dependency between signals in transit. An extreme case is sequential dependency, where the delivery date of each new signal is obtained by adding its communication delay to the delivery date of the preceding signal in the channel. Timed automata allow to express any kind of dependencies, which ones are useful is to be decided at modeling level.

Eager transitions are triggered "as soon as enabled, without letting time pass" whether they have a time dependent enabling condition or not: e.g. they can be used for modeling transitions triggered by a timeout signal to make timeout immediate, or for atomic steps taking zero time.

States with only *lazy* transitions are dangerous in real time design, as the system may never progress beyond this point. Nevertheless, they are useful for modeling a time non deterministic environment, or for modeling alternative behaviors which might or might not become possible within some time slice.

Notice that nothing more is needed than urgency and time dependent enabling conditions for expressing most concepts useful at language level:

interrupts do not need any new primitive. An SDL transition which can be interrupted just has an alternative representing the interruption in every semantic state in between its starting and its end state. As system steps are instantaneous at the semantic level, an interrupt is allowed to occur at every point of time during which the system is within the transition. Interrupts which leave the process in an "undefined" state, has to be modeled at the semantic level by means of an undefined value or non deterministic assignments. Modeling of interrupts in the current ASM semantics for SDL is more complicated than just an alternative in the semantic model.

suspension or ***pre-emption*** as used in scheduling can also be expressed without any new primitive. For this purpose, the state "*in-step*" associated with each atomic step, needs to be refined into two states "*computing*" and "*suspended*" and the transitions between them to be controlled by a *scheduler automaton*; the scheduler

guarantees that always *at most one*[3] of the to be scheduled concurrent behaviors is in the state "*computing*", and also that there is as often as possible *at least one* behavior in state "*computing*" to guarantee maximal progress.

Obviously in presence of scheduling, the enabling condition of the transition exiting each atomic step, which constrains the duration of the step, must take into account the time passed in the state "*suspended*"; that is, in Figure 2, the condition "2≤(***now***-x)≤3" should be replaced by "2≤((***now***-x) $-d_{susp}$)≤3" if the interval [2,3] constrains the execution time and not the overall duration of the step. Notice that, in timed automata, there exists the concept "*stopwatch*", a "*clock"* which can be stopped from time to time and restarted again later, and whose value is the duration since the last reset during which it has not been stopped. Thus, stopwatches can be used in an obvious manner to model durations in presence of suspension without introducing the "suspended" state explicitly in the behavior automaton (see for example [AGS00]).

The representation of *scheduling laws*, according to which the schedulers suspends and (re)starts processes, needs a means to chose the right transition according to the scheduling law. In [AGS00] shows that any scheduling laws can be represented by means of dynamic priorities, given in the form of rules of the form "$\boldsymbol{c}$ ==> $\boldsymbol{t_1 > t_2}$", meaning that whenever condition c holds transition t_1 has higher priority than transition t_2. This is not necessarily how scheduling laws should be implemented, but it defines a very general semantic scheduling framework, allowing to model any possible scheduling algorithm (such as by process priorities, RMA, EDF,..). The time needed for scheduling can, where this is relevant, be modeled by waiting in the scheduler during which no scheduled process is active.

3. Modeling non functional aspects

Modeling is usually done with the perspective of performing analysis. Modeling for analysis requires building a model of the *system* **and** of the *environment*. In the context of real time systems the environment includes time.

The expression of constraints should be convenient, abstract and different from functional notations. Standard SDL can not be used for this purpose, as modeling the time environment of the systems means imposing constraints on time spent in transitions, in communications and also in all other implicitly defined activities, and SDL has only functional timing primitives allowing to define minimal bounds on the starting time of transitions and tasks. Therefore new concepts have to be introduced, which also must be different from functional concepts.

Notice that the time semantics of current SDL tools [Geode,Tau], in strong contradiction to the standard, considers SDL transitions as *instantaneous* and *eager*, that is time progress is similar as in the underlying semantic model. This allows to express maximal bounds on time progress in states, at the price of using functional primitives, such as timeouts or time guards, to model non functional aspects. This makes the resulting specification hard to read and inadequate for code generation.

[3] or at most *n*, where n is the number of available resources

In the sequel of this section we discuss constraints expressing (known, assumed or desired) characteristics of the (time) environment, that is the underlying execution system.

The purpose is *not* the expression of *requirements,* depending on the environment *and* the system dynamic.

Communication delays

For the expression of communication delays, two questions must be answered. *What kind* of constraints are needed, and *how to associate* them with "communications". Concerning the association, there exist several possibilities:

- attach constraints with outputs, that is the "source" of communications; this has the advantage to ease the expression of signal dependent communication delays, but it introduces overheads and possible inconsistencies between delays associated with similar communications.
- as all communication is through channels, a straightforward option consists in attaching constraints with delaying channels. This option has been chosen in time enhanced versions of Geode and Tau.
- SDL channels define logical communication paths, which must later on be mapped to physical means for communication (which might be everything from a shared variable to the internet); obviously taking into account the actual target architectures (outside SDL), allows to obtain more faithful estimations of communication delays.

The last two options are not necessarily exclusive. The first one is to be used in absence of information on the target architecture. When this information is available, it is a better source for assumptions on communication delays. A sanity check between the two levels of assumptions, consists in verifying if for every possible end-to-end communication, the constraint obtained from the architecture refines the constraint defined at SDL level.

The types of useful constraints, especially for performance analysis, are manifold, and can probably not be fully captured by simple annotations as we propose them. A reasonable solution consists in:

- defining a set of annotations allowing analysis at an abstract level.
- taking into account characterizations of the communication media which can be given in the architecture description; as long as they can be used as a model of the channel behavior for the construction of the overall semantic model; SDL can be used to define such channel behaviors, where the use of functional time constructs is not problematic (as no code is generated for channel behaviors)

We consider only a small set of SDL channel annotations which we believe sufficient together with the above mentioned possibility for defining a more fine grained performance model. The choice of annotations has been made with the motivation to allow at least the features implemented in [O01] in order to take into account losses and two types of communication delays. Channels can have

- a ***loss rate***, which is defined by an expression evaluating to a real in [0,1].
- A delay – either of type ***pipeline*** or ***delay*** –defined by an expression of type "interval of duration"; an interval of durations is defined by two duration expressions representing its minimal and the maximal value.

Constraints of type "*pipeline*" are *load independent* (for modeling communication media with some degree of parallelism, such as the internet), and constraints of type "*delay*" are the sequential *load dependent* type of constraints (for modeling purely sequential media) as defined in section 2.

Restricting loss probabilities and durations to constant expressions allows static type checking and eases analysis, whereas data dependent constraints increase expressiveness. Our point of view is that, at language definition level, such constraints have not there place, but tools will introduce the constraints necessary for analysis.

Execution times

Also for execution time constraints, we have to identify the useful *types* of constraints:

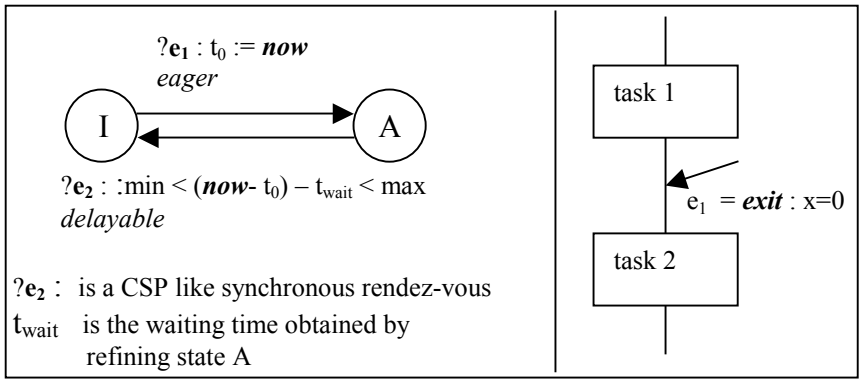
- in the context of timing analysis the assumption that everything takes an arbitrary amount of time except if specified otherwise, is unrealistic. A more realistic approach is on the contrary that system activities take zero time and no waiting in state, unless specified explicitly otherwise. Its up to the designer not to abuse of this facility.
- we consider duration constraints of type "interval of durations" defined by two duration expressions for the minimal and the maximal value of the interval
- we distinguish between
    - *absolute durations*, constraining the overall time passing between the start and the termination of a behavior,
    - and *execution times*, which do not include time for waiting for the environment or the disponibility of the processor resource.

  In models close to the implementation, the second type is more interesting, but also global estimations (abstracting away from exact waiting times for scheduling by including estimations of them) are quite frequently used. Absolute durations allow to constrain signals from the environment without necessarily modeling them explicitly. Notice that waiting for communications from *within* the system cannot be considered as included in the constraint durations as they are determined by the system itself, whereas the intention here is to express pure environment constraints,.

At the semantic level, a duration constraint on a given behavior - a sub graph of the control graph representing the functional model - are expressed by generalizing the idea expressed in Fig.2 of section 2. The fact that a duration measure is "stopped" in states waiting for the scheduler or a signal to arrive, can also be expressed exactly as already explained in section 2.

The second question is for which sort of behaviors one can constrain the duration.

- the time extended versions of SDL tools allow to associate duration constraints with tasks, outputs, decisions and any single SDL behavior constructs; this is acceptable when explicit constraints are explicitly given only for the constructs whose duration is judged non negligible. Obviously, this approach requires some rigor of the designer as zero time loops (Zeno behaviors) must not be excluded.
- it is also convenient to associate *durations* with more complex pieces of sequential behavior, such as transitions or procedures constraining the duration between start and end state(s). The case of non termination is a functional design error.
- The possibility to associate a "duration" with *agents* is restricted as it applies at best to passive agents without re-entrance.
- a more flexible way of constraining sequential behaviors which are not necessarily within a single process, is obtained by constraining the time passing between pairs of "events" (e_1,e_2) which need not to be in the same process. In the semantic model such a constraint is represented by an automaton as in the figure below, executed in parallel with the constraint behavior, "activated" on occurrence of e_1, deactivated by the occurrence of e_2 and imposing time progress by less then the minimal duration as long the constraint is active.



Possible re-entry can be handled by activating a new occurrence of the constraint automaton at each occurrence of the "start" event e_1. These "events" correspond to semantic level events of type "change" state; states are graphically represented vertical lines in an SDL transition, and two events "enter state" and "exit state" are associated with them. We therefore attach events with vertical lines and distinguish its two events by a keyword ***enter*** or ***exit***. Moreover the occurrence of an event can be constraint by a condition.

Notice that a good graphical representation for such constraints is an MSC like notation, similar as proposed in [SPI97,DDH+01], which are used for constraining time progress.

The discussed features are very interesting from the point of view of expressiveness as they allow loose time constraints avoiding over specification,. Nevertheless, analysis of systems with time constraints of behaviors containing many system interactions induce a tremendous amount of non-determinism, limiting the possibilities of analysis. Constraining only pieces of behaviors without communications, allows more compositional analysis.

Also the use of "overlapping" time constraints is not allowed as the satisfiability of the a set of overlapping constraints is undecidable, and thus the underlying semantic model.

Time constraints of the environment

For the signals arriving from the environment, one needs to specify *response times*, *inter arrival times*, *jitters*, …

In SDL there exists no particular notation for the environment and must be modeled by processes (for which no code is generated). Notice that all the above features can easily be expressed by means of *eager* or *delayable* transitions and time guards controlling the time when the environment sends a signal. As for the environment process no code is generated, it is unproblematic to use functional constructs for expressing non functional time constraints of the environment, and no special notations need to be introduced.

Notice however that a special notation for behaviors representing the environment, as the notion of "actor" in UML, would help to obtain more readable designs.

Scheduling

How to represent scheduling information in SDL, and the question if it is a good option to express scheduling within SDL, is out of the scope of this paper. We briefly discuss only the information necessary to construct the semantic level scheduler (as introduced in section°2) which controls a set of processes by means of a set of priority rules.

The information on deployment of processes on processing units, allows to define the association between the semantic level schedulers of section 2 (representing processing units) and controlled processes. Without any further information, the semantic model represents non deterministic scheduling and analysis of maximal execution times valid for "any" scheduler.

The set of priority rules allowing deterministic scheduling could be given as such by the user or calculated for standard scheduling algorithms (RMA,EDF,..) where it is important to have the information about the pre-emptibility of (sequences of) atomic steps. In [AGS00] a methodology is presented for defining the priority rules in a modular way starting with the innermost agencies; at every level one needs to relate transitions of different agents at the same structuring level.

Local time

A fundamental aspect of modern real-time distributed systems that makes them especially complex to model and reason about, is the absence of a global system clock, and thus time. Temporal synchronization between distributed components must be achieved by the system itself where the simplifying assumption that the time reference ***now*** has everywhere the same value is not appropriate. It is also often the

case that this temporal coordination of the components is the key area where SDL and its associated model checking tools should be applied, i.e. this is the most complex design area where unforeseen errors such as deadlocks, livelocks etc caused by the temporal coordination of the components, are likely to be introduced. A notion of local time or local clock is therefore important.

Local time can be expressed, but in an awkward manner: in a real time system there must always exist a defined relationship between the external reference time (***now***) and local time, defined for example by a maximal drift or offset. In this context, any condition on local time can be transformed into a (weaker) condition on ***now***, reflecting the possible values of ***now*** for a given local time value.

We proposed to introduce the notion of local time (defined by a drift and/or offset with respect to the global time ***now***); local clocks or timers progress with respect to their reference time.

4. Functional time related concepts

SDL is a modeling language, but it is also used to automatically generate code, and for this reason the functional constructs must be well distinct from non functional constructs. In its classical application domain, that is communication protocols, the existing functional features, time guards and timeouts, are sufficient as time is mainly used for time stamping and to avoid indefinite waiting[4].

Time stamping

Time stamping is quite cumbersome in SDL. At certain control points the current value "***now***" is stored in a time variable (or sent as a parameter of a signal) and at a later point the difference between the current and the stored value of "***now***" is used in functional decisions

The "*clock*" primitive of timed automata, as introduced earlier can also be used conveniently in the context of functional design for measuring durations without explicit reference to absolute time ***now***. Thus, clocks do not increase expressivity, but are a methodological help for the designer to use absolute time ***now*** in an inappropriate way.

Interrupts

During the entire execution of an SDL level transition, the concerned process is insensitive to signals from the environment. In order to increase the reactivity of a process it is convenient to have the possibility of *interruption*.

---

[4] E.g., for a signal that has been lost or has never been sent because a part of the system has failed silently

The real-time language Esterel [BG92,Esterel] has a primitive ***watchdog*** for exiting a transition on an external signal. State-charts offer in addition the possibility of "*deep history re-entry*" which allows to return to the point of control where the transition has been exited, that is to explicitly model *pre-emption*.

In SDL, the *possibility of* pre-emption can be obtained by using concurrent processes for the pre-empting and the pre-empted transition, as *concurrency* leaves the room open for arbitrary interleaving of atomic steps of the transitions at execution time, that is any possible *pre-emptive behavior*.

interruption by signals or time progress can not be expressed in SDL in a convenient way, as they imply reactivity to the environment within a transition. An inelegant, and often used, workaround consists in

- testing time at various places within the transition if interrupt can be caused by time progress
- cutting the transition into several transitions allowing to test the arrival of an interrupt signals more often.
- When even a single task may take longer than the allowed reaction time to the interrupt, this task is often modeled by a slave process, started by the master process which waits for the slave process to terminate and remains always reactive to an interrupt signal.

All these workarounds emulate in more or less precise way at SDL level the underlying semantic model; the first two methods imply an undesirable discretisation and to unreadable specifications and the third one induces a considerable modeling overhead and still does not allow to terminate a time consuming activity which has become useless for some reason. The introduction of an explicit "interrupt" primitive leads clearly to a greater modularity of the design. We propose to use an extension of the exception mechanism to represent interrupts. As it has been shown in section 2, the translation of this primitive in the semantic model is straightforward.

Time under the control of the system

The assumption that, in the implementation, time is external to the system is appropriate in most systems. Nevertheless, whenever a clock synchronization algorithm is designed, or whenever the "system clocks" have to be considered "part of the system" rather than external, it is important to have primitives which on one hand are closely related to external time, but are under the control of the system. A *clock* as those used in timed automata introduced above, can play this role, as it increases with time and can be set by the system to any specific value allowing corrections of clock values. The recently accepted Profile for Real-time, Scheduling and Performance [RFP02] and [MSD+01] propose slightly different "clocks" (in the sequel called *UML clocks*). They also depend on external time, but instead of offering a value whenever they are asked for, they are by essence discrete and send periodically a *time signal* containing the current value of the clock. Thus, in terms of SDL, they are cyclic timers sending the time value attached with the "timer" signal. As well the timed automata clocks as the UML clocks can be used to model system clocks – which have to be used instead of "*now*" in all functional constructs. Notice that the timed

automata clocks are more abstract and avoid the explicit consumption of time signals when time is not needed.

5. Conclusion

This paper motivates a number of time concepts necessary for modeling functional and non functional time related aspects of systems and shows how they can be expressed in the semantic model.

The main aim of the paper is to underline the necessity of strict distinction between primitives for functional and non functional modeling, and to propose a rich set of primitives for modeling non functional aspects, in particular time constraints, so far completely lacking in the standard.

We exclusively consider time constraints representing the *environment* of the system, that is entities outside the system which exchange signals with the system, and the underlying execution platform which determines the execution time. The constraints are designed in such a way to disallow the expression of mixed constraints on both the environment and the system.

This is the purpose of *requirements* to be expressed in an requirement language. In ObjectGeode, requirements can be expressed by observers. MSC are too weak to express general time dependent safety properties. But extensions of MSC, such as Live sequence charts [DH99], can be good candidates for a requirement language.

In this paper we have not discussed the introduction of an appropriate atomicity: to allow compositional modeling, a sequence of steps can be considered as *atomic* if interaction with the environment takes place only at starting time and/or at termination time of the sequence (atomic steps are represented at the semantic model by two transitions, thus guaranteeing a single interaction in each transition). Interactions with the environment are sending or reception of signals, but also RPC, and use of ***now*** in decisions,… Thus individual transitions (and sometimes even the evaluation of time dependent expressions) must be cut into several atomic steps, resulting in a very fine grained semantic model. Especially, the use of ***now*** within transitions is problematic, as whatever processes are grouped into a "module", reading time is always a communication with the environment. Fixing the choice of the value of ***now*** for at least a part of the transition leads to more reasonable granularity (and implementation).

Concerning performance analysis, we don't believe that very fine grained timing analysis (taking into account time for individual assignments, evaluation of single expressions, …) are appropriate at SDL level, as the distance between an SDL model and the generated code (where several processes may be grouped into a single thread, sending of signals become assignments,…and moreover code optimization induced important changes) is too important. But, good analysis results can be obtained by "measuring" execution times of pieces of behavior for a given implementation on a given platform and back annotating them into an SDL model for analysis. The tool proposed in [CPP*01] allowing timing analysis of Esterel programs is based on this idea.

Finally, the relationship with the Profile on Performance scheduling and Time of OMG [OMG02] which has been recently accepted, should be clarified. The aim of this profile is to provide a catalogue of notations – aiming almost for exhaustivity - and the definition of their interdependencies. There is very little about semantics or how these notations could be used for analysis. The aim of the work presented in this article is on the contrary the definition of a minimal set of notations necessary for design and analysis and to provide a precise semantics. Nevertheless, conformance with the notations defined in the UML profile, whenever this is reasonably possible, is preferable. Some, but not all notations and concepts have already been defined with this profile in mind.

6. References

[HNSY92] T.A. Henzinger, X. Nicollin, J. Sifakis, S. Yovine, "*Symbolic model checking for real-time systems", Annual Symposium on Logic in Computer Science*", IEEE Computer Society Press, 1992

[AH94] R. Alur, T.A. Henzinger, "*Real-time system = discrete system + clock variables", Theories and Experiences for Real-time System Development*", AMAST Series in Computing 2", 1994

[ALH95] B. Algayres, Y. Lejeune, F. Hugonnet, "*GOAL: Observing SDL behaviors with Geode*". Proceedings of SDL Forum 95.

[ADL+99] J-M Alvarez, M. Diaz, L. Llopis, E. Pimentel, J.M. Troya, "*Integrating Schedulability Analysis and Design Techniques in SDL*", Real Time Systems Journal

[ALV99] Alvarez J.M, Diaz M., Llopis L. M., Pimentel E., Troya J. M. "*Embedded Real-time Systems Development using SDL*", IEEE Real-time Symposium, 1999

[ADL+01] J-M. Alvarez, M. Diaz, L. Llopis, E. Pimentel, J.M. Troya., "*Deriving Hard-Real Time Embedded Systems Implementations Directly from SDL Specifications*", CODES'01: 9th International Symposium on Hardware/Software Codesign. 25-27 April

[AGS00] K. Altisen, G. Gössler and J. Sifakis. "*A Methodology for the Construction of Scheduled Systems*" FTRTFT 2000, Pune, India, September 2000, LNCS 1926, pp.106-120.

[BG92] G. Berry, G. Gonthier. "*The Esterel synchronous programming language: design, semantics, implementation*". In Science of Computer Programming, 19(2), Elsevier, 1992

[BST98] S. Bornot, J. Sifakis and S. Tripakis, *Modeling Urgency in Timed Systems*, International Symposium: Compositionality - The Significant Difference, Malente (Holstein, Germany), 1998, LNCS Vol. 1536

[BS97] S. Bornot and J. Sifakis, *Relating Time Progress and Deadlines in Hybrid Systems,* International Workshop, HART'97, Grenoble, LNCS, Vol. 1201, 1997

[BGS00] S. Bornot, G. Gössler and J. Sifakis. "*On the Construction of Live Systems*". TACAS 2000, LNCS 1785

[BB93] F. Bause, P. Buchholz "*Qualitative and quantitative analysis of timed SDL specifications*"

[BLL*98] J. Bengtsson, K. Larsen, F Larsson, P. Pettersson, W. Yi C. Weise. "*New Generation of UPPAAL*". Int. Workshop on Software Tools for Technology Transfer. Aalborg, Denmark, July, 1998

[BFG*99] M. Bozga, J.Cl. Fernandez, L. Ghirvu, S. Graf, J.P. Krimm, L. Mounier, *IF: An Intermediate Representation for SDL and its Applications*, Proceedings of SDL-Forum'99 (Montreal, Canada), Elsevier

[BGM01] M. Bozga, S. Graf, L. Mounier, *Automated validation of distributed software using the IF environment*, 2001 IEEE International Symposium on Network Computing and Applications (NCA 2001)

[BGM*01] M. Bozga, S. Graf, L. Mounier, I. Ober, J-L. Roux and D. Vincent. *Timed Extensions for SDL*. Proceedings of SDL Forum 2001, LNCS 2078, June 2001.

[BGK00] M. Bozga, S. Graf, A. Kerbrat, L. Mounier, I. Ober and D. Vincent. *SDL for real-time: what is missing?*. Proceedings of SDL & MSC Workshop. Grenoble, June 2000.

[CPP*01] E. Closse, M. Poize, J. Pulou J. Sifakis, P. Venier, D. Weil, S. Yovine "*TAXYS : a Tool for the Development and Verification of Real-Time Embedded Systems*", CAV 2001

[CS01] J.-L. Camus, T. Le Sergent, "*Combining SDL with Synchronous Dataflow modeling for distributed Control Systems*" SDL Forum 2002, LNCS 2078, June 2001

[CNRS88]. "*Le temps-réel*" TSI - Technique et Science Informatiques 7(5): 493-500

[D21GEN02] "*Timed extensions for SDL*" Delayed ITU Document, February 2002

[DDH+01] M. Dörfel, W. Dulz, R. Hofmann, and R. Münzenberger: "*SDL and non-functional requirement"*. Internal Report IMMD7 05/01, University of Erlangen-Nuremberg, Germany, August 20, 2001.

[DH99] Damm W., Harel D "*Breathing live into sequence charts*" Formal Methods in System Design, 2001

[EGG*00] R. Eschbach, U. Glässer, R. Gotzhein, A. Prinz, "On the Formal Semantics of Design Languages: A compilation approach using Abstract State Machines". Proc. ASM 2000, Int. Workshop on Abstract State Machines, Switzerland, March 2000

[Esterel] Esterel Suite and Scade development tool set, see http://esterel-technologies.com/

[Geode] *ObjectGeode 4-1 Reference Manual*. Telelogic Technologies Toulouse. See also http://www.telelogic.com

[Gur97] Y. Gurevich "Draft of the ASM Guide", Technical Report University of Michigan, 1997

[HHK01] Th. Henzinger, B. Horowitz, Ch. M. Kirsch, "*Giotto: A Time-triggered Language for Embedded Programming*", 1st Int Workshop on Embedded Software (EMSOFT '01), Lecture Notes in Computer Science 2211, Springer-Verlag, 2001

[Leu95] Stefan Leue "*Specifying Real-time requirements for SDL specifications*" proceedings of PSTV'95

[MIT99] Mitschele-Thiehl A., Slomka F., "*A methodology for hardware/software co-design of realtime systems with SDL/MSC*" Computer Networks 31, 1999

[MSD+01] R. Münzenberger, F. Slomka, M. Dörfel, R. Hofmann. *A General Approach for the Specification of Real-Time Systems with SDL*. In R. Reed and J. Reed (Eds.), *Proc. of the 10th International SDL Forum*, Springer LNCS 2078. 2001.

[Mal99] M. Malek "*PerfSDL: Interface to protocol performance analysis by means of simulation*" Proceedings of SDL Forum 99

[OCK00] I. Ober, B. Coulette, A. Kerbrat "*Timed SDL Simulation and specification*", technical report Telelogic Technologies Toulouse, 2000

[O01] , I. Ober *Spécification et Validation de Systèmes Temporisés avec des Langages de description formelle: étude et mise en œuvre* (en english), Phd Thesis, Toulouse, 2001

[OMG02] *Response to the OMG RFP for Schedulability, Performance, and Time*, OMG document number: ptc/2002-03-02.

[QSDL] M. Diefenbruch, E. Heck, J. Hintelmann, and B. Müller-Clostermann. "*Performance evaluation of SDL systems adjunct by queuing models*" Proc. of SDL-Forum '95, 1995.

[R96] Rick Reed: "*Methodology for Real Time Systems*". Computer Networks and ISDN Systems 28(12): 1685-1701 (1996)

[Rou98] J.-L. Roux "*SDL Performance analysis with ObjectGeode*", Workshop on performance and time in SDL, 1998

[SDM+00] F. Slomka, M. Dörfel, R. Münzenberger, R. Hofmann. Hardware/Software Codesign and Rapid-Prototyping of Embedded Systems. *IEEE Design & Test of Computers, Special issue: Design Tools for Embedded Systems*, Vol. 17, No. 2, April-June 2000.

[SPI97] Spitz S;, Slomka F., Dörfel M. "*SDL* - an annotated specification language for engineering multimedia communication systems*" Workshop on high speed networks, Stuttgart, October 1999

[Stan88] J. A. Stankovic, "*Misconceptions about real-time: a serious problem for next generation systems*." IEEE Computer 21(10): 10-19

[Tau] TAU toolset documentation, see http://www.telelogic.com

[Z100] Recommandation Z100; Specification and Description Language (SDL). ITU, Standardization sector, November 1999