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 modelling of non functional aspects and analysis, and those needed for functional design. We allow the distinction between functional and non functional timing aspects 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 for the development of real-time and embedded systems, where the functional behaviour is time dependent.

SDL is a modelling language in which sufficient details can be given for the generation of code preserving all the properties of the specification, including those concerning time dependent behaviour. SDL has already some important time related features, such as a notion of *global time* (allowing to measure durations throughout the system by means of appropriate time stamps), and the possibility to allow time dependent decisions in the functional design (*timeouts* and time dependent *enabling conditions* allow to define constraints on the triggering time). Explicit means to quit a current activity, that is *interrupts*, and means to describe systems where time is (partly) under the control of the systems, that is *clocks,* are needed.

We call (purely) non-functional aspects those which exclusively restrict the time extended model. The time restrictions of such a model are due to hypothesis or knowledge on the environment or the underlying execution systems. Modelling such non-functional aspects of systems, such as execution times of tasks or sequences of tasks, for analysis, can not be done so far with 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], and some work on timed verification [OCK00] and schedulability [ALV99, ADL+99, ADL+01]. A general real

*This work has been partially supported by the European projects IST-1999-11557 Interval and IST-2001-33522 Omega

time framework for SDL is presented in [SDM+00] and [BGK00, BGM*01], where the first one is mainly for hardware software co-design.

Most of these approaches, in particular the ones mainly concerned with 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 and therefore not to be relevant) 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] allow to attach explicit timing information with SDL constructs such as tasks, and to provide some deployment information.

Our approach [BGK00,BGM01, ITU02] is an extension of these latter approaches, where we focus on timing properties rather than on performance[1]. In section 2, we discuss how language level real-time concepts can be expressed by means of a small set of primitives at the semantic level which we express in terms of timed automata with urgency [BST98]. Concerning the needs for real-time primitives at SDL level, as introduced in [ITU02], we distinguish between annotations for 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 primitive concepts of the specification language, where we are interested in time related primitives for SDL. . Notice that in semantic level models, time is the object of modelling and can be constraint in various ways.

An interesting semantic framework is that of timed automata [HNSY92, AH94, BST98, AGS00, BGS00], where

- time progress and system progress are along orthogonal dimensions, such that system transitions are timeless (corresponding to instantaneous events) and time progresses in system “states” only
- the system can restrict time progress in states (by means of an urgency attribute associated with transitions)
- and system transitions can be enabled or disabled by time progress.

Timed automata have a notion of global state, 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 can be considered both as a semantic level formalism and modelling language itself. The ASM model has no

---

[1] Notice that the approach can be adapted to performance evaluation in an almost straightforward way by using constraints of probabilistic nature. The main problem is that only for memoryless probability functions, tractable analysis methods exist (see [H00])

predefined time concept attached with it, but is expressive enough to express almost any time semantics. The 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, discrete transitions leading from some state to a next state, where all intermediate "micro steps", if they exist, do not appear in the model (the granularity concerns mainly the functional model, and is not the object of the discussion in this paper)
2. it does not enforce orthogonality of time and system progress; on the contrary, it forces time progress to happen in transitions rather than in states, such that an atomic step is a combined system and time progress (where time progress is allowed to be zero, but there is no time progress without a system state change.

These constraints on time and system progress are not mainly due to deficiencies in the ASM model, but to the SDL semantics itself[2]. Controllable time and orthogonality of time and system progress are fundamental concepts at the semantic level, and as timed automata with urgency [BST98, AGS00, BGS00] are based on these concepts, we use them here as an intuitive semantic modelling formalism

Obviously, for any ASM model with time progress in transitions, can be defined an equivalent one with the additional constraint that time does not progress in system transitions and conversely. Every *atomic step* is split into at least two *instantaneous atomic steps*, one or several time steps and an instantaneous system step and (see the figure below, where we represent time progress in each of the states by a single "time progress transition"; this transition may be cut into as many steps as needed in the global system).

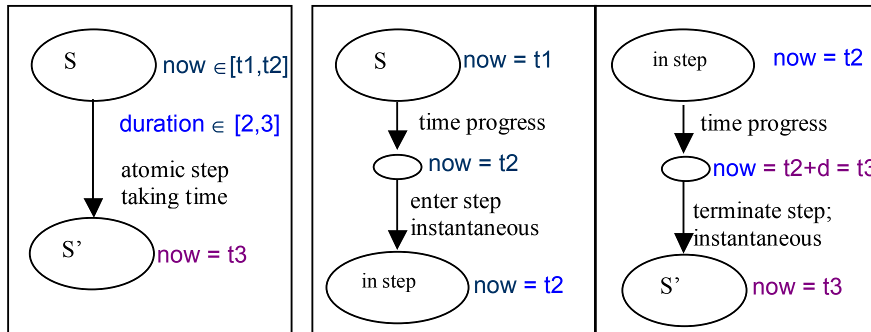


Fig. 1. transforming an atomic step taking time into two time progress transitions and two instantaneous transitions

[2] Nevertheless, the ASM model has some difficulties to express continuous time; it cannot express continuously changing entities in a compositional manner, as time progress can only be represented by an explicit state change in which time progresses by *some value.* For a realizable system, there exists always a discretisation of time allowing to represent all equivalence classes of possible behaviours, but this discretisation can not be chosen for a subsystem without knowing its context, that is the global system in which it is going to be embedded.

Timed automata, and modelling languages based upon them such as [BFG*99,BGM01, BLL*98], provide for measuring time progress a primitive called "*clock*" instead of 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 explicit use of the absolute system time ***now***. Here, we use in all examples the notation using ***now***, which from the point of view of expressiveness is equivalent.

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 suppose to be encoded in explicit states and transitions
– a time dependent enabling condition, expressing at which time points 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 (always by maintaining the same system state) 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 in a state, only one of the enabled (instantaneous) system transitions are possible, and this as long as any *eager* transition is enabled in the reached state.
  - *delayable* transitions *can wait*, but only until the falling edge of their time dependent enabling condition is reached, 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, imposes the following restriction on time progress in **s**: at time point **t**, waiting is possible for a duration **δ**, as long as the **T**ime **P**rogress **C**ondition

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

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

An atomic step which starts to be executed at time t_1, and which has a duration of 2 to 3 time units, is modelled 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, leading to the stable state S2, is triggered.

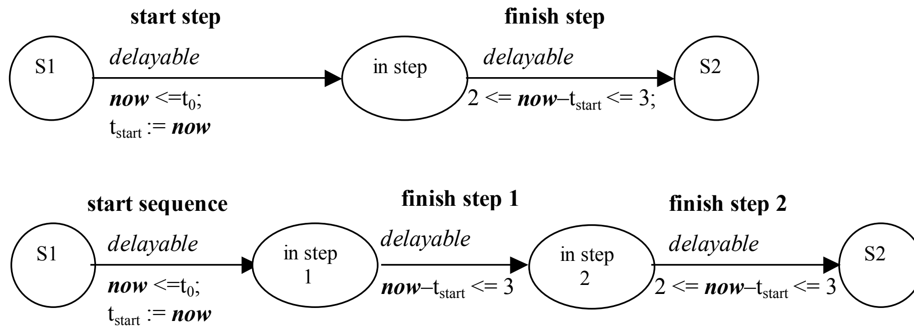


Fig.2: representing an atomic step with time constraints

The duration of a sequence of steps can be constrained in a similar way as shown in Fig.2[3].: the intermediate steps can occur at any time, but not later than the time corresponding to the maximal overall duration has passed, and the last step must additionally make sure that the overall sequence of steps takes at least the required minimal duration. This does not exclude the sequence where the duration has reached its maximal value already in the starting state *s1*; just time will not progress any further until the end of the sequence. This is 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 nonzero delay can be modelled

- By a timed automaton, receiving signals at the time they are sent, and inserting them into the input queue of the receiver process at any time point at which the time constraint on the communication delay holds
- or as in the 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.

A potential problem of the second model appears when the receiver does not yet exist at sending time, and therefore also not its input channel, but does exist at arrival time.

Both semantics allow in a straightforward manner to represent variants of channels, in which messages can be lost, reordered,….

From a modelling 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 the delays of all signals in transit. An extreme case is sequential dependency, where the delivery date of each

[3] 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.

new signal is obtained by adding its communication delay to the delivery date of the preceding signal in the channel. As well ASM as timed automata allow to express any kinds of dependencies, which ones are useful is to be decided at modelling level. The complexity of validation can be measured in the number of clocks necessary in the corresponding timed automaton model: notice that independent delays need one clock per signal in transit, whereas a "sequential" channel can be modelled with a single clock, independently of the number of signals in transit.

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 modelling transitions triggered by a timeout signal to make timeout immediate, or, when considering a discrete time abstraction, for modelling of consecutive steps taking place in the same discrete "time step".

States with only *lazy* outgoing transitions are dangerous in real time design, as the system is allowed to never progress beyond this point. Nevertheless, lazy transitions are useful for modelling a time non-deterministic environment, or for modelling alternative behaviours which might or might not happen within a given time interval.

Notice that nothing more is needed than urgency and time dependent enabling conditions for expressing most concepts useful at the semantic 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 modelled by instantaneous events with the meaning "step terminated", 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, are modelled by means of an undefined value or non deterministic assignments.

suspension or ***pre-emption*** due to scheduling can also be expressed without any new primitive. For this purpose, the state "*in-step*" associated with each step, needs to be refined into two states "*computing*" and "*suspended*". The transitions between them can either be controlled by some *scheduler automaton* which should guarantee mutual exclusion, and also that there is as often as possible a behaviour in state "*computing*" in order to guarantee maximal progress. An alternative means is to use a hierarchical system of priorities as in [Sif02] which allows to replace an explicit scheduler in a very simple manner.

Obviously when taking into account scheduling, the enabling condition of the transition exiting each atomic step, which constrains the duration of the step, must allow to take into account the time passed in the state "*suspended*"; that is, in Figure 2, if the interval [2,3] constrains the execution time and not the overall duration of the step, the condition "$2 \leq (\textbf{now}\text{-x}) \leq 3$" should be replaced by "$2 \leq ((\textbf{now}\text{-x}) - duration_{susp}) \leq 3$". 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 behaviour 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 “ $\boldsymbol{c} \implies \boldsymbol{t_1} > \boldsymbol{t_2}$”, meaning that whenever condition c holds transition t_1 has higher priority than transition t_2. This is not necessarily how scheduling laws will be implemented, but it defines a general and compositional semantic scheduling framework, allowing to model any possible scheduling algorithm.

3. Modelling duration constraints in SDL

Modelling 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. Standard SDL is very cumbersome and can not really be used for this purpose: modelling the time environment of the systems means expressing constraints on time spent in transitions, in communications and also in all other implicitly defined activities by means of timers or time guards.
But timers and time guards allow only the specification of minimal durations. Notice that, the time semantics of SDL tools [Geode, Tau], in strong contradiction to the standard, considers SDL transitions to be *instantaneous* and *eager*, and time progresses *exclusively* where timers or time guards impose *waiting*. That means, time progress is similar as in the underlying semantic model and maximal time bounds can be expressed.
Systematic modelling of durations by means of explicit time guards, leads to an inappropriate interdependence between functional design and time constraints. It makes the resulting specification hard to read and inadequate for code generation, as time constraints are not to be interpreted in a unique way when it comes to the generation of code.
We discuss here constraints expressing (known, assumed or desired) timing characteristics of the environment and the underlying execution system.

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:

- as all communications are through channels, which may imply some positive delay or not, a straightforward option consists in attaching constraints with delaying channels. This option has been chosen in time enhanced versions of Geode and Tau and is well accepted by users.
- 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 possible communication delays.

These 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 description 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 communication constraints for time and 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 characteristics of the communication media given in the architecture description

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 modelling 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 modelling purely sequential media) as defined in section 2. "Mixed channels", where the communication delay is partially load dependent, can be obtained by the sequential composition of channels of the two delay types.

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 restrictions of the subset of allowed expressions have not their place, but tools will define them depending on their analysis capabilities.

Processing times

The processing of a signal in an agent is done in several phases. From the point of view of the external "user", we are interested in *response time* constraints, which is divided into queuing time and treatment time; the treatment time itself consists of pure execution time and blocking time, during which it waits to be scheduled or to get some response to a subordinate task. A set of non-overlapping constraints are

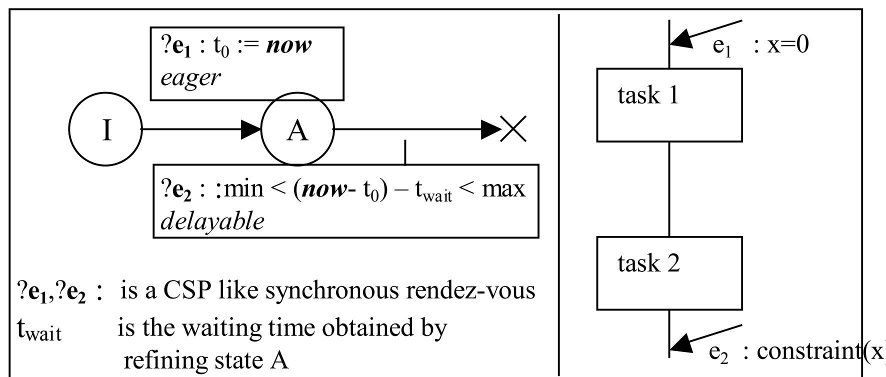
sufficient as global constraints can be derived from them. Morover, non overlapping constraints make analysis easier.

We consider duration constraints of type “interval of durations” defined by two duration expressions for the minimal and the maximal value of the interval
As already mentioned, it is interesting to distinguish between

- *delay*, constraining the overall time passing between the start and the termination of a behaviour,
- and *execution time*, which does not include time for waiting for the environment or the availability 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. Durations allow to constrain signals from the environment without necessarily modelling them explicitly.
At the semantic level, a duration constraint on a given behaviour - 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 behaviours 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 behaviour constructs.
- it is also convenient to associate *durations* with more complex pieces of sequential behaviour, such as transitions or procedures constraining the duration between start and end state(s). The case of non termination is then a functional design error.
- The association of a “duration” with *agents* can be useful if all its activities are subject to the same time constraints.



- a more flexible way of constraining sequential behaviours which are not necessarily within a single process, is obtained by constraining the time passing between arbitrary pairs of “events” e_1 and 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 functional behaviour, “activated” on occurrence of e_1, inactivated by the occurrence of e_2 and imposing time progress by less then the minimal duration as long the constraint is active. Possible re-entrance 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, that is vertical lines in SDL specifications. The set of time constraints implies conditions on the occurrence time of all these events.

Notice that timed MSC allow already exactly this type of constraints for scenario specifications, and in [SPI97,DDH+01] MSC like notations are used for for expressing constraints on processes.

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 on behaviours containing many system interactions induce a tremendous amount of non-determinism. Constraining only pieces of behaviours without communications with the exterior of a given agent allows more compositional analysis.

The use of “overlapping” time constraints is problematic as the satisfiability of the a set of overlapping constraints is a hard synthesis problem, and thus the construction of the semantic model.

Execution Modes

An interesting question is how to interpret parts of the system on which no time constraints are expressed. The SDL semantics suggests to consider them as “lazy”, that is time may pass arbitrarily. In the synchronous approach, however, the designer would on the contrary make the assumption that all non time constraint actions are immediate. In order to offer a flexible framework with different possible execution modes, the explicit introduction of urgency of not explicitly time constraint parts, at the user level, would be interesting.

Time constraints on the external environment

For the signals arriving from the environment, *response times*, *inter arrival times*, *jitters*, … are non overlapping constraints and must be expressible.

The environment will be modelled by processes (for which no code is generated) and all the above features can be expressed by means of time guards

In a component based approach however, one wants to verify a subsystem in by using the above mentioned timing characteristics of its environment.

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 only discuss briefly the information necessary to construct the semantic level scheduling rules as mentioned in section 2.
The information on deployment of agents on processing units, allows to define the association between the semantic level schedulers of section 2 (representing processing units) and controlled processes..
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 preemptibility of (sequences of) atomic steps. In [AGS00] a methodology is presented for defining the priority rules in a compositional and hierarchical manner starting with the innermost agents 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, live-locks 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 allows the description of time dependent functional behaviours by means of timeouts and time stamping. Timeouts allow the triggering of an action when "some delay has been exceeded" whereas time-stamps are more expressive and allow to define triggers, decisions and timers depending on all observed durations (as far as they can be expressed by means of (local) time stamps).

These 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. In the application domain of communication protocols, the existing functional features, time guards and timeouts, are mainly used to react when some response (signal) is taken into account too late or to avoid indefinite waiting[4].

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*.

For example in Esterel [BG92,Esterel] exists 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 modelling the pre-empting and the pre-empted transition in concurrent processes, as *concurrency* leaves room for any sequentializations of atomic steps of the transitions at execution time, that is any possible *pre-emptive behaviour*.

Interruption by either signal reception or time progress can not be expressed in SDL in a convenient way, as this implies 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 a transition into several transitions allowing to test the arrival of an interrupt signals more often.
- modelling time consuming tasks in 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 modelling 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, time, represented by some system clock, is external to the system, is appropriate in most systems. Nevertheless, whenever a clock

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

synchronization algorithm is designed, or the "system clocks" are "part of the system" rather than external, it is important to have a primitive corresponding to a partially controllable system clock. C*locks* as those used in timed automata introduced above, can play this role, as they increase depending on (external) 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 such "clocks"; with the difference that they are intrinsically discrete (that is, their value changes with some well-defined granularity). Both, timed automata clocks and discrete UML clocks can be used to model system clocks – which are used instead of "*now*" in all functional constructs. Nevertheless, timed automata clocks, which need not to be a priori discretized, are more abstract.

5. Conclusions

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

Functional time related primitives are those which allow explicitly define alternative behaviours depending on observed time. Non-functional primitives express timing features orthogonal to the functional behaviour, and they consist in constraints on the (relative) occurrence time of events, and are completely lacking in the standard. In this paper we have not discussed the introduction of an appropriate *atomicity*. To allow compositional modelling, 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 and tasks 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, good analysis results can be obtained by "measuring" or calculating bounds for execution times of pieces of behaviour for a *given implementation* on a *given platform* and back-annotating them into an (abstract) SDL model for analysis. The tool proposed in [CPP*01] allowing timing analysis of Esterel programs is based on this idea.

The relationship with the profile on "Performance, scheduling and time" of OMG [OMG02] which has been recently accepted, remains to be clarified. This profile provides mainly a catalogue of notations and notations 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 here, is precisely the definition of a minimal set of notations, necessary for design and analysis and to provide a precise

semantics. Conformance with the notations defined in the UML profile, whenever this is reasonably possible, is ongoing work, for example in the IST project OMEGA[5]

6. References

[AHP96] R. Alur, G. Holzmann, D. Peled, "*An analyser for message sequence charts*", TACAS'96, Lecture Notes in Computer Science Vol 1055, 1996

[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.

[BL97] H. Ben-Abdalla, S. Leue, "*Expressing and analysing timing constraints in message sequence chart specifications*", technical report, U. of Waterloo, 1997

[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.

[5] see http://www-omega.imag.fr

[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
[ITU02] "*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, 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, LNCS 2211, 2001
[Leu95] Stefan Leue "*Specifying Real-time requirements for SDL specifications*" 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