How useful is the UML profile SPT without Semantics?[1]

Susanne Graf, Ileana Ober

VERIMAG
2, avenue de Vignate - F-38610 Gières - France
e-mail:{Susanne.Graf, Ileana.Ober}@imag.fr
http://www-verimag.imag.fr/~{graf,iober}

The UML Profile for Scheduling, Performance and Time (SPT) defines a set of concepts useful for modeling real-time systems. Its purpose is to integrate notation used by existing real-time analysis techniques into UML in order to

1. Enable the construction of models that could be used to make quantitative predictions regarding these characteristics.
2. Facilitate communication of design intent between developers in a standard way.
3. Enable inter-operability between various analysis and design tools." (pp. 1-1 of SPT)

Thus, the SPT is defined to offer a common framework for real-time modeling that unifies the diversity of techniques, terminologies and notations existing in the real-time software community, while still leaving space for different kinds of specifications. In its present form, the main focus of SPT is on time and time-related concepts: performance, timelines, schedulability, etc, where for example with respect to schedulability mainly the existing standards theories (RMA) are covered.

To answer these goals, the SPT offers a terminology for modeling real-time systems: it defines a set of concepts - aiming to fit standard real-time modeling techniques - and some relationships between these concepts as allowed by the meta-modeling technique used for the definition of the SPT. Clearly, these meta-modeling techniques can carry only very superficial semantic information. At a first sight this may be argued by its aim to address the needs of various real-time modeling techniques. However, a closer look to the SPT definition itself shows that such a definition is insufficient, in particular for promoting common understanding of specification and exchange of specifications between different tools, which is, at least for a large number of users, an important motivation for switching from proprietary formalisms and tools to standards, like UML.

1. Our thesis: the existence of semantic information provides more freedom

The thesis that we develop in this position paper is that providing semantic information is the only way to both make the initial goal of the SPT of providing a communication framework between developers and tools become real, and of preserving the freedom of tools builders to add and adapt concepts depending on the needs of particular applications or application domains.

The meta-modeling technique used for defining the SPT has the advantage that it offers a lightweight definition of the concepts contained in the profile and it shows some of their mutual dependency. The SPT definition uses the same technique as the one used for the definition of the UML and the UML notions that are needed or affected by the SPT definition are present in the SPT. This way the relationships between the notions contained in SPT, and the UML notions are made clearer, as well as the relationship of the SPT itself with the UML.

One of the ideas of the SPT is to provide concepts by means of *keywords*, by taking care to use already existing widely used terminology: e.g. it introduces concepts such as *timer*, *clock*, *response time, WCET* (worst case execution time), ... This leads inevitably to two types of problems:

- Different communities use different names for the same concept. As an example take the term *timer* and *clock*: which have the opposite meanings depending on the communities in which

[1] The work on which this position paper is founded has been carried out in the IST OMEGA project and in the IST ARTIST initiative.

they are used. The existing meta-modeling techniques allow easily the exchange of models between tools (and humans) using different names for the same underlying concepts. So this problem is solved.

- But for any concept used in a given community and application domain can in general not be directly mapped into a concept of another domain. Even if globally the used concepts are similar there are often subtle differences depending on the different needs.
  For example, let's take a *time*r: setting a timer provides some *trigger* after a given delay (or at a given point of time) which will be consumed by the object which has set the timer. There are at least 3 different restrictions (in fact many more) on the trigger consumption which are useful and are indeed used in different frameworks: (1) a trigger that must be consumed immediately or is lost, (2) a trigger that *will* be consumed immediately and (3) a trigger that can be stored (in the object's event queue) for later use.
  The situation is similar for notions like *ResponseTime*: from the point of view of the caller, the definition of response time is relatively unambiguous, but from the point of view of the callee there are several interpretations, for example, it may or may not include the queuing time in the callees input queue.
  Also the notion of *TimedEvent* is not free of ambiguity: it can be attached to any UML model element and specifies "the time of the start of the associated behavior". Now, if one attaches a *TimedEvent* to a state machine state with entry states, the actual meaning of the time event is not obvious: is it the entry action start time, is it the entry action end time, or is it something else?

The approach used in SPT to deal with the variability of concepts is to add attributes in the form of keywords and to abandon the idea of fixing a semantics (leave it up to the tools) – and thus to abandon the initial goals. Indeed, fixing a semantics has the inconvenient that there will always be some domains in which slightly different concepts are needed and developers have as much difficulties as today to talk to each other.

The main question is: *how to provide a flexible and relatively open framework and still be able to exchange models with their semantics?*

A commonly proposed solution to this problem is that, indeed the UML meta-model does not provide semantic information, but in a MDA approach, *model transformation* is the key issue, and the model transformers are an integral part of the model for defining the semantics. Indeed, just like for any programming language, the compiler together with the execution platform provides a semantics (which needs not to be completely deterministic). But does this help the users and the tool implementers? It means that the users must try to understand the semantics by studying the transformation program or play with an interactive simulation tool, which is the situation which we have exactly today. It does also not provide any high level information for helping a user to understand a model coming from a given tool, or to help a tool provider to write a transformation for models using the concepts of tool ***A*** into a model using the concepts of tool ***B***, if the concepts of these tools do not match exactly.

The only real solution to overcome this problem is to provide a standard way to describe semantic information at a high level of abstraction. For the general dynamic semantics, the main issues are the choice of the granularity and communication and execution mechanisms, that is, the possible choices between several concurrently enabled steps and the granularity of the observed steps. The number of reasonable communication modes is relatively small, and in particular in the context of SPT, an effort has been made to provide attributes with widely accepted interpretation. Concerning the restriction of the inherent potential concurrency, a wide variety of choices exist, and different choices are made in different tools: some consider an UML model as an (almost) deterministic specification by imposing a particular ordering, whereas others leave some possible choices open, for leaving room for refinement at a later stage.

Extending a behavior of a system with time consist in attaching time points with instances of state change, where a priori the only restriction is that causally ordered state changes (or events) are also (at least weakly) ordered with respect to time. If we are able to provide a means to *name* all relevant state changes (this are the *TimedEvents*), and to capture the time elapsing between any two occurrences of such state changes (captured by *duration expressions*), then the concepts in SPT, as well as any variations of them, can be defined by means of duration expressions and constraints on them (e.g. in the form of a subset of OCL constraints, another standard within UML). The semantic definition of timers and clocks include a behavioral part (see for example [1]), but notice that their semantic subtleties stem from the differences in the time constraints on the events associated with them.

In the next section, we show how the approach that has been followed in the OMEGA project can be used to make step further towards a high level semantic definition of UML profiles, in particular SPT.

2. Our approach

The object of the IST project OMEGA [3] is to provide a framework for the development of correct real-time systems based on the use of an UML profile and the use of formal method based validation tools. In this context it is important to have a well-defined semantics, possibly defining certain choices as non-deterministic, as validation tools provide useful answers only if they interpret the meaning of the studied model in the intended way. In OMEGA, we have defined a particular semantic profile for the functional behavior (see [4], [5]), and we have defined a real-time profile (see) that can be regarded as a specialization of the SPT, but our interpretation of the usefulness of this profile is that of being an illustration of how to define a profile allowing the semantic definition of any particular timing profile at an abstract level. The main ingredients of our timing profile are:

- The domains *time* and *duration*. We could provide axiomatizations of these types in the form of axiomatic abstract data types, but it is sufficient to consider them as sub-domains of real (or natural numbers) with a restricted set of operations (time points can only be compared, time can only grow,….).

- We define a notion of *timedEv*ent. An important aspect is here that we associate with syntactic concepts (methods, signals, timers, actions, state machine states, ...) and a given context (class, state machine, transition, ...) one or several events: e.g., for a state machine state an *entry* and *exit* event is defined, for a signal communication *send*, *receive* and *consume* events are defined, where the first is attached to its sender and the two others to its destination. An event definition can also include a condition on the values of its parameters or of the current state allowing the distinction of events depending on dynamic information. Moreover, events can *store* state information: this allows extracting from the entire state some abstraction useful for timing. This extension of *TimedEvent* is needed if one aims at defining timing constraints depending exclusively on the information provided by a set of well-defined timed events defining an abstraction for timing analysis.

The dynamic interpretation of any *instance* of a *timedEvent* is a sequence of occurrences (with monotonically increasing occurrence times). One can syntactically refer to different occurrence of an event instance *E,* by referring to the current occurrence *E,* or to previous ones in the form of *E.pre*, *E.pre.pre*, *...* [2]

- We define several expressions of the form $SomeDuration(E_1,E_2)$ defining the time distance between well defined occurrences of specified events. This allows for example to define the semantics of a *ResponseTime* associated with some method call as the distance between the corresponding occurrences of the events *acceptinvoke* and *sendreturn* associated with this method call, thus defining it as a response time seen from the callee without accounting for the queuing time. Finally, notions like WCET as defined in SPT become then simple constraints on duration expressions. For example "*WCET=p*" is expressed by a constraint of the form "$duration(E_1,E_2) \leq p$".

- We syntactically distinguish two types of constraints, assumptions and requirements. An *assumption* defines a restriction on the defined model, that is, only executions satisfying all assumptions are valid executions whereas requirements are properties which must be inferable

[2] We prefer this over the access via event indexes to simplify the expressions, especially when relating of occurrences of different events which need not to have indexes increasing at related speeds

from the model: a model satisfies a requirement when all valid sequences satisfy also all requirements. This distinction is not only important for timing constraints.

- Timers and clocks are defined by implicit active objects and behaviors defined in terms of state machines. These state machines define the functionality of timers (e.g. each *set* event will be followed by a *reset* of *timeout* event) and are part of the semantic definition. Constraints concerning timing, i.e. the fact that the *timeout* event occurs exactly at the time point defined by the *set* action, or that the transition triggered by the timeout must be triggered immediately or within some delay, can be expressed by constraints on the durations between the different events associated with a timer (*set*, *expire*, *consume*,...).

The timing profile is structured into two layers: primitive and derived. Primitive timing extensions correspond to basic notions as defined above where the semantic definitions are given by means of easy-to-understand OCL-like constructs. Derived concepts, corresponding essentially to (a subset of) the terminology as defined in SPT, are defined via a transformation into primitive concepts as shown above. No additional formalism to express this semantics is needed. The advantage is that the model of a real-time system expressed in UML with timing information expressed using the constructs contained in our profile is self-contained.

Concerning the dynamic semantics of a functional UML model, we have started by defining a particular profile which is based on so-called activity groups (similar to Room capsules) which treat requests in run-to-completion manner, but otherwise concurrent activities are considered a priori as unordered (see [4,5]). With the intention to provide a list of possible semantics, we have started to study the set of all possible semantic variation points of the OMEGA subset of UML [7]. In [6], we study a general interaction model providing all communication modes used in practice (synchronous call, asynchronous signal, asynchronous call ...) as a particular instance. The toolset defined in is based on the translation from UML into an intermediate representation where some of the UML level concepts are mapped into more primitive concepts. The particular execution model defined in Omega is captured by means of a small number of high level priority rules (also studied in [6]). Changing or refining this profile or a particular model, can be obtained simply by adapting the set of priority rules. We use this in practice for the description of scheduling constraints. The practical experiences with such rules have been made so far at the level of the intermediate representation (where some rules are defined implicitly by the underlying semantics), but we believe that the same approach could be used also directly at the level of UML.

3. Conclusions

We see a risk that the SPT becomes a marketing buzzword, and fails to meet its primary goals of facilitate communication of models and the interchange between tools. The current trend of taking care of the multiplicity of concepts used in different application domains by the introduction of specialized syntax in the form of tags or stereotypes which are then interpreted by a particular tool makes it de facto impossible to interpret a given model otherwise than with the (unique) appropriate tool and compromises the value of UML as a formalisms for exchanging models between different kinds of users.

The need for a means to define UML profile also semantically is real; especially in the context of real-time and embedded systems, where dynamic properties of models are the key issue. This is needed for the exchange of models between designers and for tool builders to communicate the particularities of their tools. Providing a full formal semantics for a rich set of notations like UML is not what we are aiming for. We are rather addressing the problem of *explaining semantic differences* between a set of relatively close approaches agreeing on a large number of concepts and their meaning. Only in such a context, exchanging models does make sense, and we argue that given some general underlying semantic model, a framework for the definition of the particularities of a given profile would be both useful and achievable. In the Omega project, we have shown a way to go into this direction. In order to define the semantic subtleties of the Omega profile we argue that this can be done without using an external formalism, by means of OCL-like constructs and easy to understand priority rules.

References

[1] Susanne Graf, Iulian Ober, Ileana Ober. *Timed annotations with UML*. In Proc. of workshop on Specification and Validation of UML models for Real Time and Embedded Systems, SVERTS, associated with UML 2003, technical report Verimag 2003/10/22, October 2003

[2] Iulian Ober, Susanne Graf, Ileana Ober. *Validating timed UML models by simulation and verification*. In Proc. of workshop on Specification and Validation of UML models for Real Time and Embedded Systems, SVERTS, associated with UML 2003, technical report Verimag 2003/10/22, October 2003

[3] Susanne Graf, Jozef Hooman, *The Omega project: Correct Development of Embedded Systems*. In Proc. of European Workshop on Software Architectures, EWSA, associated with ICSE 2004, LNCS, 2004

[4] W. Damm, B. Josko, A. Pnueli, A. Votintseva, *Understanding UML: A Formal Semantics of Concurrency and Communication in Real-Time UML*. Proc. of FMCO'02, November 5–8, 2002, Leiden, the Netherlands, LNCS Tutorials 2852.

[5] M. van der Zwaag, J. Hooman, *A Semantics of Communicating Reactive Objects with Timing*. In Proc. of Workshop on Specification and Validation of UML models for Real-Time Embedded Systems, SVERTS associated with UML 2003, technical report Verimag 2003/10/22

[6] G. Gößler, J. Sifakis. *Composition for Component-Based Modeling*. Proceedings of FMCO'02, November 5–8, 2002, Leiden, the Netherlands, LNCS Tutorials 2852, pp 443-466.

[7] H. Fecher, M. Kyas, F. de Boer. *Variation points of the UML semantics concerning methods*. In Deliverable D.1.1.4 on Time extensions and semantics of the IST OMEGA project