Softw Syst Model (2004) 3: 181–193 / Digital Object Identifier (DOI) 10.1007/s10270-003-0045-7

Special section on graph transformations and visual modeling techniques

Dynamic Meta Modeling with time: Specifying the semantics of multimedia sequence diagrams

Jan Hendrik Hausmann, Reiko Heckel, Stefan Sauer

University of Paderborn, Institute for Computer Science, D 33095 Paderborn, Germany
E-mail: {hausmann,reiko,sauer}@upb.de

Received: 7 February 2003/Accepted: 14 May 2003
Published online: 1 April 2004 – © Springer-Verlag 2004

Abstract. The Unified Modeling Langugage (UML) offers different diagram types to model the behavior of software systems. In some domains like embedded real-time systems or multimedia systems, it is necessary to include specifications of time in behavioral models since the correctness of these applications depends on the fulfillment of temporal requirements in addition to functional requirements. UML thus already incorporates language features to model time and temporal constraints. Such model elements must have an equivalent in the semantic domain.

We have proposed Dynamic Meta Modeling (DMM), an approach based on graph transformation, as a means for specifying operational semantics of dynamic UML diagrams. In this article, we extend this approach to also account for time by extending the semantic domain to timed graph transformation. This enables us to define the operational semantics of UML diagrams with time specifications. As an example, we provide semantics for special sequence diagrams from the domain of multimedia application modeling.

Keywords: Formal semantics – Meta modeling – UML extensions – Graph transformation – Time – Multimedia – Sequence diagram

1 Introduction

The key objective of modeling is to create a representation of reality or ideas that abstracts from unnecessary details and concentrates on the main concepts. When designing software systems, time aspects are often (although sometimes unreasonably) considered a minor requirement and are thus not represented in the models. Consequently, the core diagrams of UML [18] – the standard language for building visual models of software systems – focus on structure, function, and dynamics of systems, but not on temporal aspects.

While this approach is adequate for example in the construction of business software (where temporal requirements typically concern efficiency, which mostly depends on the underlying hardware, system software, and database systems), temporal behavior is a key feature in other domains, and it has to be represented in the model in a precise way. Embedded real-time systems and multimedia applications are the most prominent among these domains. Real-time systems require that the results of a computational task are available within a limited period of time. In multimedia applications, timing requirements especially refer to the processing and synchronization of continuous media objects and the related quality of service (QoS).

The UML already provides some syntactic elements to express temporal behavior, like send and receive times of messages and duration of intervals on sequence diagrams or firing times for transitions in statecharts. These elements may be used to formulate timing constraints, i.e., time expressions on stimuli, message, or transition names. Further elements are being introduced by UML profiles like the *UML Profile for Schedulability, Performance, and Time* [17], which is motivated by the domain of embedded real-time systems. Its temporal modeling more fundamentally deals with time and time values, time-related events and stimuli, timing mechanisms like timers and clocks, and timing services.

Yet, while the semantics of the frequently used core elements of the UML is only partly understood, the interpretation of time-related modeling concepts in UML is even more ambiguous. The real-time profile [17] adds some detail in this regard, but still lacks a precise and formal semantics.

We can thus identify both a strong need for the precise specification of temporal behavior and a lack of concepts in the UML to meet this demand.

Existing approaches for real-time system specification are mainly motivated by the need to analyze (i.e., test or verify) systems with respect to their fulfillment of temporal properties. These approaches generally use time constraints to prescribe temporal requirements for a system. These constraints can be modeled in UML sequence diagrams. The operational execution of a system is rather described by an automata-based model, e.g. a state machine with timed events. It can then be tested or verified whether the state machine model or an implementation conforms to the timing constraints specified in the sequence diagram. Examples for this can be found in [13] where statecharts are extended by information of worst-case execution times derived from an actual implementation. Other approaches check whether an implementation satisfying all constraints may actually exist by using model checkers [1, 5] or systems of linear inequalities [14]. [1] defines an operational semantics of a real-time extended subset of UML statecharts by translating them to Uppaal timed automata [15] and model-checking them.

In contrast to the aforementionend translation approaches, we present an approach to the operational semantics of UML diagrams in this article that resides on a higher level of abstraction, disregarding the need to be familiar with mathematical formalisms or model-checker languages. It incorporates a notion of time, thus enabling precise interpretation of models with temporal information. The approach consists of specifying an abstract interpreter for the behavioral diagrams of interest. For this purpose, diagrams are represented as instances of a meta model (i.e., as object diagrams, formally regarded as attributed graphs) which extends the UML meta model by representations of runtime state information. The steps of the interpreter are specified by graph transformation rules which manipulate the runtime state information to model the execution of the diagram. This approach to Dynamic Meta Modeling (DMM) has been successfully applied to statechart and sequence diagrams [2, 9]. Note that the use of graph transformation rules is different from the graph-grammar based transformation presented in [19]. There, UML models that are annotated with performance information (like execution time) are translated into a stochastic performance model by means of graph-grammar productions.

In order to account for the time aspect, we extend the formal foundation of the DMM approach from attributed to timed graph transformation systems [7]. Consequently, the approach is called *Dynamic Meta Modeling with time* (*DMM+t*). Following a formal introduction to the approach in Sect. 2, Sect. 3 presents – as a case study for DMM+t – multimedia sequence diagrams. They are a specialization of UML sequence diagrams for modeling multimedia applications and have been proposed as part of the OMMMA approach towards object-oriented modeling of multimedia applications based on UML (see [4, 20] for details). The operational semantics of these diagrams is defined using DMM+t in Sect. 4, where we also show how the formal semantics can be employed to gain additional information on the example under consideration.

A careful review of the achieved results in Sect. 5 reveals that in special cases the semantics definition yields results that do not correspond to the intuitive concepts. Two different kinds of strengthening the semantics are introduced that can be used to eliminate these unwanted phenomena. Section 6 concludes the presentation and points out possibilities for future work.

A preliminary version of this work has been presented at the International Workshop on Graph Transformation and Visual Modeling Techniques (GTVMT 2002) in Barcelona [10].

2 Dynamic Meta Modeling with time

While textual programming languages are defined by means of grammars and abstractly represented by terms or trees, the UML is defined by its meta model [18], i.e., a class diagram augmented with constraints, whose instances represent individual UML models. Interpreting these instances as attributed graphs, it is natural to use graph transformations to specify the manipulation and execution of diagrams. The approach of Dynamic Meta Modeling (DMM) [2, 9] uses rule-based graph transformations, denoted as UML collaborations, to specify abstract interpreters for dynamic sub-languages of the UML, and thus provides an operational semantics.

In order to provide semantics to modeling techniques with time, like multimedia sequence diagrams, the representation of time in graph transformation systems has been studied in [7]. Instead of introducing time as a separate semantic concept, the approach models time by means of time-valued attributes representing logical clocks. This bears the advantage that different aspects and properties of time can be modeled, depending on the strategies according to which time values are assigned and updated.

2.1 Graph transformation

Dynamic Meta Modeling with time (DMM+t) is based on typed and attributed graphs which are represented as UML class and object diagrams. Graph transformation is defined according to the algebraic approach, which is given a set-theoretic description in [7]. We denote rules by $p : L \to R$ and transformation steps by $G \overset{p(o)}{\Longrightarrow} H$, where p is the rule and o its occurrence, and consider sequences $G_0 \overset{p_1(o_1)}{\Longrightarrow} \cdots \overset{p_n(o_n)}{\Longrightarrow} G_n$ of transformations up to permutation of independent steps.

To be more precise, a notion of equivalence is defined on transformation sequences which considers two sequences as equivalent if they can be obtained from each other by repeatedly swapping independent transformation steps. This equivalence has been formalized by the notion of *shift-equivalence* [12], and it is based on the notion of *independence* of graph transformations: two transformations $G \overset{p_1(o_1)}{\Longrightarrow} X \overset{p_2(o_2)}{\Longrightarrow} H$ are *sequentially independent* if the occurrences $o_1(R_1)$ of the right hand side of p_1 and $o_2(L_2)$ of the left hand side of p_2 do only overlap in objects of X that are preserved by both steps, formally $o_1(R_1) \cap o_2(L_2) \subseteq o_1(L_1 \cap R_1) \cap o_2(L_2 \cap R_2)$. Otherwise, there exists a *causal dependency* between the two steps forcing their application in the given order: either the match $o_2(L_2)$ of the second step contains vertices or edges created by the first step, or the second step removes vertices or edges that have been part of the match $o_1(L_1)$ of the first.

Two alternative transformations $G \overset{p_1(o_1)}{\Longrightarrow} H_1$ and $G \overset{p_2(o_2)}{\Longrightarrow} H_2$ are *parallel independent* if the occurrence $o_1(L_1)$ of the left hand side of p_1 in G is preserved by the application of p_2, and vice versa. Otherwise the two steps are *in conflict*.

While sequential independence allows consecutive transformations to be swapped, parallel independence allows alternative transformations to be scheduled in any order with the same result. The semantic idea behind these notions is expressed in the local Church–Rosser Theorem [3].

2.2 Graph transformation with time

Next we review the basic concepts of graph transformation with time, following [7]. To incorporate time into graph transformation with attributes, we generalize the approach of TER nets [8]. TER nets are high-level Petri nets that model time as a token attribute. Therefore, a time data type is required as a domain for time-valued attributes.

We model time by means of logical clocks represented by special attributes of a *time data type* $T = \langle D_{time}, +, 0, \geq \rangle$, i.e., an algebraic structure where $\geq$ is a partial order with 0 as its least element, $\langle +, 0 \rangle$ forms a monoid (that is, + is associative with neutral element 0), and + is monotonic wrt. $\geq$. Obvious examples include natural or real numbers with the usual interpretation of the operations, but not dates in the YY:MM:DD format (due to the Y2K problem).

A *graph with time* over a given time data type T is a graph in which all vertices are attributed with a special attribute *chronos* of type T. This attribute represents the state of the local clock of the object. Graph transformation rules with time $p : L \to R$ are just pairs of graphs with time as introduced above that respect the particular properties of time. This is expressed in the following axioms.

1. Local monotonicity: for all vertices $x \in L$ and $y \in R$: $x.chronos \leq y.chronos$, and
2. Uniform time stamps: for all vertices $x, y \in R$: $x.chronos = y.chronos$.

These axioms ensure a behavior of time which can be described informally as follows: according to axiom 1 an operation or step specified by a rule cannot take negative time, i.e., it cannot decrease the chronos values of the nodes it is applied to. It is, however, permitted to take zero time. If this option seems too idealistic, the zero case can be excluded without affecting the results of this article.

Axiom 2 states an assumption about atomicity of rule application, that is, all effects specified in the right hand side are observed at the same time, called the *firing time* of the rule application. Hence, it is guaranteed that the chronos value of each object always represents the last point in time when the object took part in a rule application (thus the last time it had an externally visible behavior).

In this case, one can show in analogy with TER nets that for each transformation sequence s using only rules that satisfy the above two conditions, there exists an equivalent sequence s' such that s' is time-ordered, that is, time is monotonically non-decreasing as the sequence advances. Thereby we obtain the behavior of a fully synchronized system with global time by strictly local means.

Theorem 1 (global monotonicity [7]**).** *For every transformation sequence s using only rules that satisfy axioms 1 and 2 above, there exists an equivalent sequence* $s' = G_0 \overset{p_1(o_1),t_1}{\Longrightarrow} \dots \overset{p_n(o_n),t_n}{\Longrightarrow} G_n$ *such that* s' *is time-ordered, that is,* $t_i \leq t_{i+1}$ *for all* $i \in \{1, \dots, n-1\}$.

The abstract interpreter specified in this manner by a set of graph transformation rules with time is mathematically represented as a rewrite relation over instance graphs $G \overset{p(o),t}{\Longrightarrow} H$ labeled with occurrences of rules $p(o)$ and their firing times t. In addition, we specify a set of *terminal instance graphs* to distinguish successful termination from deadlock. Then, a *trace of the interpreter* is a sequence of transformation steps ending in a terminal state corresponding to a terminal instance graph. Since we do not want to distinguish different interleavings of concurrent actions, we consider such traces up to *shift-equivalence*. Theorem 1 ensures that one time-ordered representative exists in every equivalence class of traces.

The rewrite relation defined above also induces a notion of equivalence on graphs: two graphs are equivalent if they are reducible to the same sets of terminal graphs. (Note that there may be more than one terminal instance graph reachable from a given graph because the rewrite relation is, in general, non-deterministic.) This equivalence can be used to define a notion of semantic equivalence on UML diagrams, even if they are syntactically different.

We use this model in Sect. 4 to specify an abstract interpreter for multimedia sequence diagrams. Given a set of individual diagrams as input to this interpreter, we can test under which conditions a scenario executes successfully (by reaching a terminal state) and whether two given scenarios are equivalent (if they always produce equivalent traces or end up in the same terminal states).

3 Specifying time in multimedia with UML

Temporal relationships between elements of media presentations are the key characteristics of multimedia applications. The behavioral model of an interactive multimedia application has to account for both the timed and synchronized rendering of predefined scenes and the alteration of the course of presentation caused by user interaction. In the OMMMA approach, we deploy *multimedia sequence diagrams* (in the following: MM sequence diagrams), which are extended UML sequence diagrams, to model the former and UML statecharts to model the latter (the details of these modeling views and their integration can be found in [20]). Within this article, we concentrate on the representation of time in MM sequence diagrams to explain the DMM+t approach to formal semantics of UML with time.

We choose an example from a cinema application to illustrate our approach. In a cinema, typically there is a break to sell ice cream before the feature movie starts. We model this situation as a scene using an MM sequence diagram (see Fig. 1). While the ice cream is being sold, an advertisement slide is presented for 200 seconds. A sound clip announcing new products or special offers (Intro) followed by some "appetite-inducing" music is being played while the vendors sell the ice cream. The intro does not have a fixed length since it is subject to frequent change; the background music can be played indefinitely. The music is stopped when the movie is about to start.

The special elements used in this kind of multimedia modeling can be explained in natural language:

- All objects appearing at the top of a MM sequence diagram are *application objects*. They have the ability to render the content of some kind of media object. Every application object contains the methods start and stop which control the rendering of the media. Additional methods for pausing and re-synchronization purposes are not used in this article. Application objects are independent in their timekeeping, following the principles of distributed multimedia systems [16]. That means neither a central controller nor a global clock can be assumed.
- The boxes on the lifelines of the application objects are *presentations*. These elements replace the UML construct *activation*. A presentation represents the rendering of a media element by an application object. The name of the media object is given inside of the presentation's box. A presentation may furthermore define constraints on the minimal and maximal length of the media rendering. This means that shorter media elements would remain visible/audible even though their duration is over (e.g. static media elements like the slide have a duration of 0, but need to be shown for some time) and that long (possibly infinite) media objects (e.g. streams) can be limited. These features are needed because a scene (as described by the MM sequence diagram) does not require all media elements to have a known and fixed duration. MM sequence diagrams can thus be compared to higher-level and role-based interaction diagrams that are typically provided in the analysis phase of a software development.
- Attached to a presentation are incoming and outgoing messages. Incoming messages aligned with the top of a presentation box are messages calling the predefined method start (startmessages), incoming messages aligned with the bottom of a presentation box are stopmessages. Outgoing messages start or stop other presentations in synchronization with the current presentation. Outgoing messages can be synchronized either with the start of a presentation (starting at the top of the sending presentation box), the end of the presentation (starting at the bottom of the box), or they are sent out with a certain delay after the start of the presentation (starting at the side of the box with the specification of the delay attached). For instance, the message to start the audio playback in the example is timed to happen 5 seconds after the presentation of the slide started.

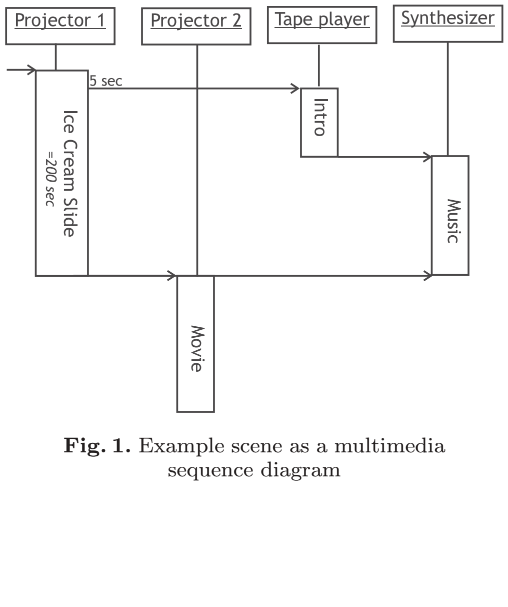


Fig. 1. Example scene as a multimedia sequence diagram

In Sect. 4.2 these concepts will be formalized using DMM+t rules. Like every semantics definition they are based on the abstract rather than the concrete syntax of the language. Therefore, the meta model defining the new language elements for MM sequence diagrams is given in Fig. 2.

The most prominent new feature in the meta model is the data type Time. This data type may contain positive integer values and the special value unlim denoting an unlimited time. It is used throughout the meta model to specify timepoints (e.g. Message.timestamp or Presentation.endtime) or the length of time intervals

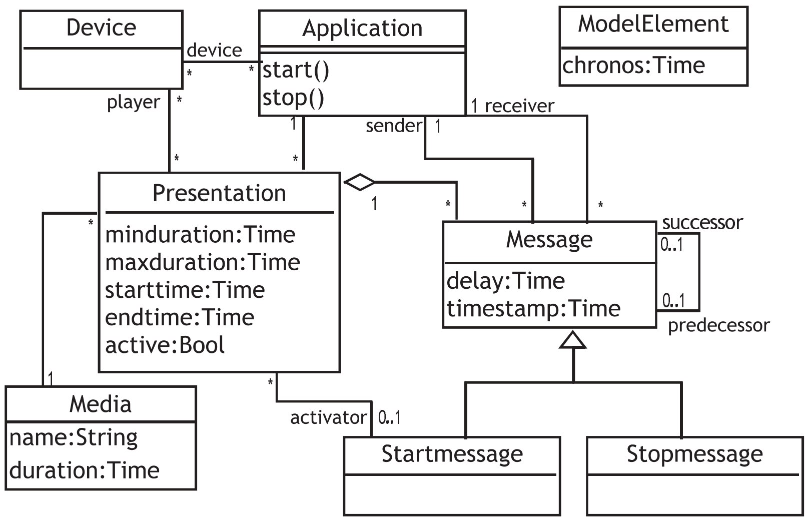
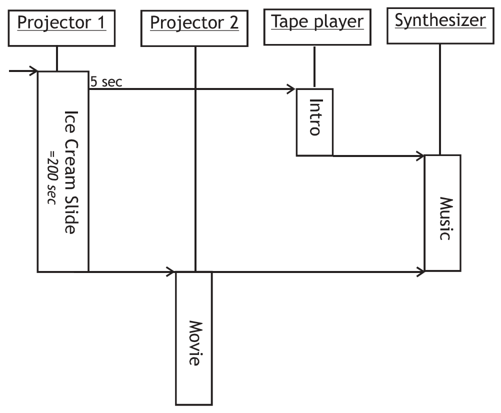


Fig. 2. Meta model of the MM sequence diagram



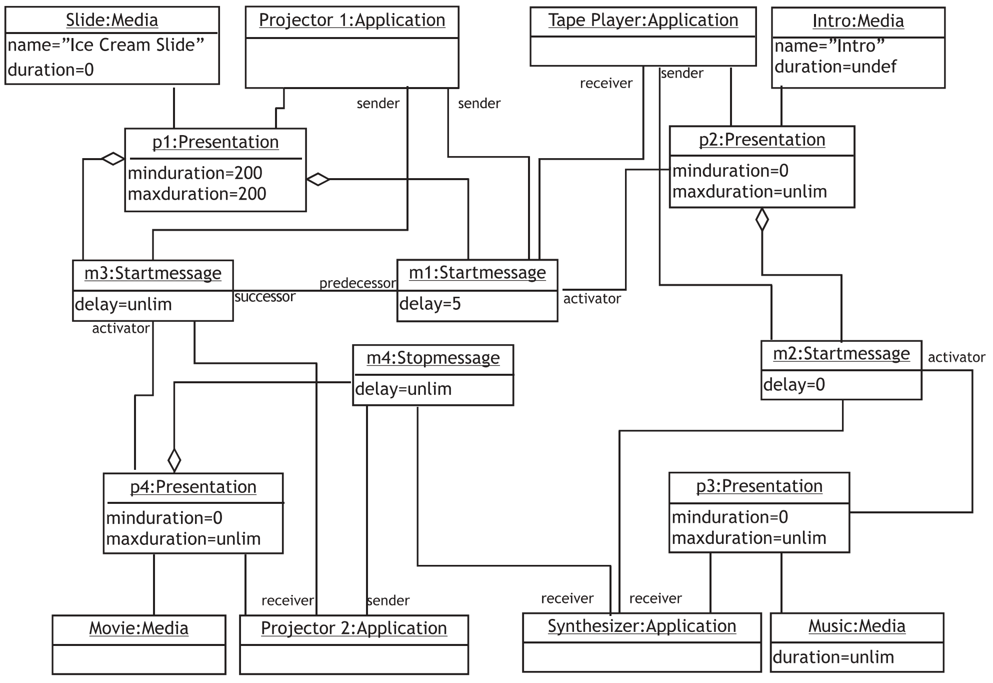


Fig. 3. Abstract syntax of the example diagram

(e.g. Media.duration). As every model element must have a chronos attribute (to comply with the axioms), it was added to the topmost class of the UML meta model hierarchy ModelElement. While a presentation uses the symbol of the UML element activation (which has no representation in the abstract syntax), it corresponds to the meta model element Presentation that contains the attributes minduration and maxduration. The associations between the incoming and outgoing messages in standard UML have consequently been adapted to this new element. The activator is now the message that starts a presentation, and all resulting outgoing messages are owned by the presentation. The order of these messages is still determined by the successor/predecessor relation. If message delays are specified, they must not contradict this order. Attributes like presentation.active or message.timestamp represent runtime information necessary for the interpretation of the operational semantics rules. A Device is the representation of a physical rendering facility suitable for the media type (this is not elaborated here).

The representation of the example diagram of Fig. 1 according to this meta model is shown in Fig. 3. Additional information in the meta model representation is the duration of the media objects (formerly only given in the text) and the assumption of default values for constraints of presentations: minduration is by default 0, maxduration is by default unlim. Everything else is just a representation of the information present in the concrete syntax diagram of Fig. 1.

Next we show how the techniques formally introduced in Sect. 2 can be used to formalize the semantics of MM sequence diagrams.

4 Semantics of multimedia sequence diagrams

This section contains the DMM+t rules for the MM sequence diagram example. The section consists of three parts: first, we introduce the rule notation as it appears in the figures, then we define the set of DMM+t rules for MM sequence diagrams, and finally we show how these rules support a precise interpretation of the model.

4.1 Format of the DMM+t rules

DMM+t rules as introduced in Sect. 2 are represented as pairs of UML collaboration diagrams on the level of classifier roles (in contrast to the instance level) where the role name following the slash symbol is optional and only shown if it is needed for binding and the name of the base classifier follows the colon symbol in the name string (see [18, pp. 3–124ff]). The collaboration diagrams are extended by attribute conditions (given in OCL). The left hand side diagram denotes the pre- and the right hand side diagram the post-conditions of the transformation. Since all rules here should conform to the axioms defined in Sect. 2, we introduce a special variable firingtime. This variable represents the chronos value that is associated with the rule's invocation. Its possible values are defined in the preconditions section of the rule. The execution mechanism of the rule has to ensure that

– the chronos attributes of all objects on the left hand side of the rule have a lower or equal value to that of firingtime before applying the rule, and
– the chronos attributes of all objects on the right hand side of the rule are updated to the value of firingtime after the rule has been executed.

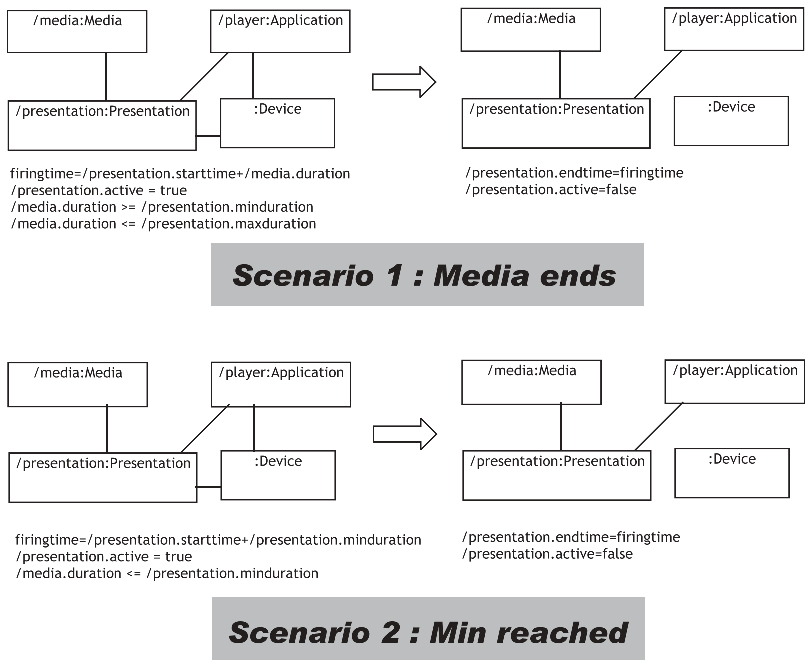

/media:Media
/player:Application
/presentation:Presentation
:Device
/media:Media
/player:Application
/presentation:Presentation
:Device
firingtime=/presentation.starttime+/media.duration
/presentation.active = true
/media.duration >= /presentation.minduration
/media.duration <= /presentation.maxduration
/presentation.endtime=firingtime
/presentation.active=false
Scenario 1 : Media ends
/media:Media
/player:Application
/presentation:Presentation
:Device
/media:Media
/player:Application
/presentation:Presentation
:Device
firingtime=/presentation.starttime+/presentation.minduration
/presentation.active = true
/media.duration <= /presentation.minduration
/presentation.endtime=firingtime
/presentation.active=false
Scenario 2 : Min reached

Fig. 4. DMM+t rule for ending a presentation due to the media duration

Specifying the firing time in this fashion is a convenient way to ensure that all resulting rules are correct with respect to the axioms. The value of firingtime may also be used in other conditions to set timestamps etc.

4.2 DMM+t rules for multimedia sequence diagrams

The DMM+t rules defining the semantics of MM sequence diagrams can be distinguished in three groups: rules that describe the end of a presentation (Figs. 4 to 6), rules that describe the reception of a message (Figs. 7 and 8), and rules that describe the sending of messages (Figs. 9 to 11).

One of the features of MM sequence diagrams that requires a specification in a formal time-based semantic domain is the end of a presentation. This end may occur in a number of ways. Either the media duration is inside the specified parameters and the presentation stops "naturally" (i.e., the presentation runs just as long as the media's duration), or the minduration or maxduration constraints force the end of the rendering to occur at a certain point in time. A further possibility is the reception of a stopmessage. Figures 4 to 7 specify these alternatives. The simplest one is the normal end of a media object. The corresponding rule is represented in Fig. 4. The preconditions state that this rule is only applicable if media.duration fulfills the specified constraints, i.e., if it falls inside the specified interval. Note that the association (association role, to be precise) to the :Device role is only present on the left hand side of the rule, i.e., it is deleted when applying the rule since the device is deallocated. This is the same for all rules ending a presentation. The two rules in Figs. 5 and 6 specify the end of the presentation due to the minduration or maxduration con-

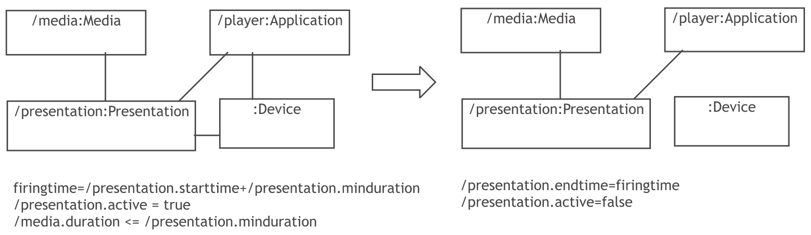

/media:Media
/player:Application
/presentation:Presentation
:Device
firingtime=/presentation.starttime+/presentation.minduration
/presentation.active = true
/media.duration <= /presentation.minduration
/media:Media
/player:Application
/presentation:Presentation
:Device
/presentation.endtime=firingtime
/presentation.active=false

Fig. 5. DMM+t rule for extending a presentation up to minduration

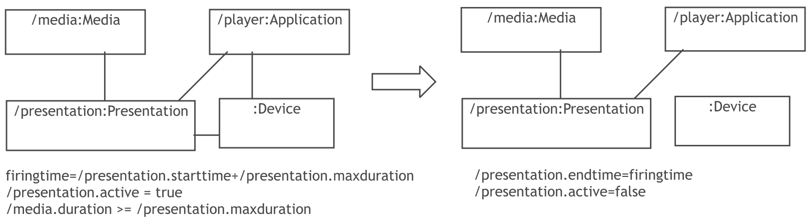

/media:Media
/player:Application
/presentation:Presentation
:Device
firingtime=/presentation.starttime+/presentation.maxduration
/presentation.active = true
/media.duration >= /presentation.maxduration
/media:Media
/player:Application
/presentation:Presentation
:Device
/presentation.endtime=firingtime
/presentation.active=false

Fig. 6. DMM+t rule for cutting a presentation at maxduration

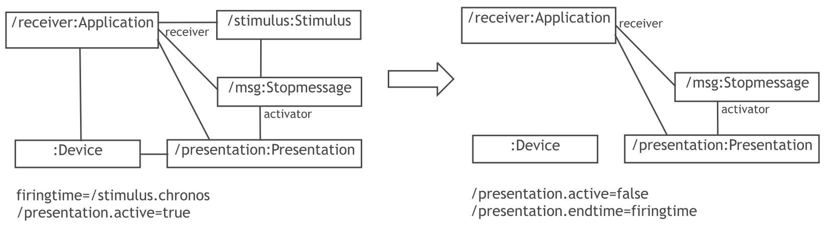

/receiver:Application
receiver
/stimulus:Stimulus
/msg:Stopmessage
activator
:Device
/presentation:Presentation
firingtime=/stimulus.chronos
/presentation.active=true
/receiver:Application
receiver
/msg:Stopmessage
activator
:Device
/presentation:Presentation
/presentation.active=false
/presentation.endtime=firingtime

Fig. 7. DMM+t for receiving a stopmessage

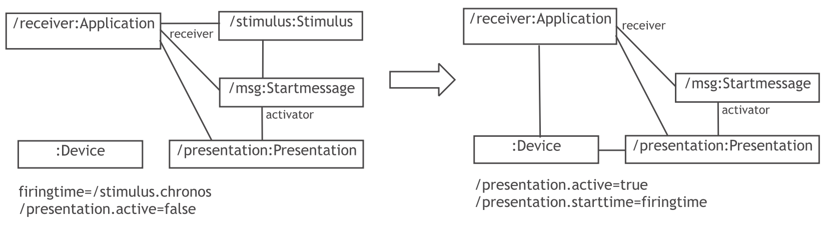

/receiver:Application
receiver
/stimulus:Stimulus
/msg:Startmessage
activator
:Device
/presentation:Presentation
firingtime=/stimulus.chronos
/presentation.active=false
/receiver:Application
receiver
/msg:Startmessage
activator
:Device
/presentation:Presentation
/presentation.active=true
/presentation.starttime=firingtime

Fig. 8. DMM+t rule for receiving a startmessage

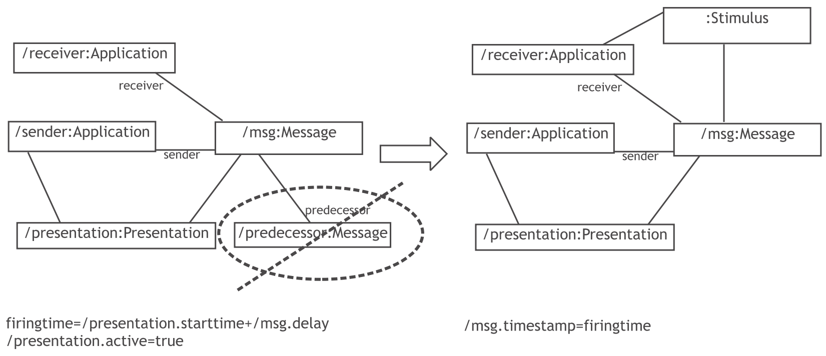


Fig. 9. DMM+t rule describing the sending of the first message in an order

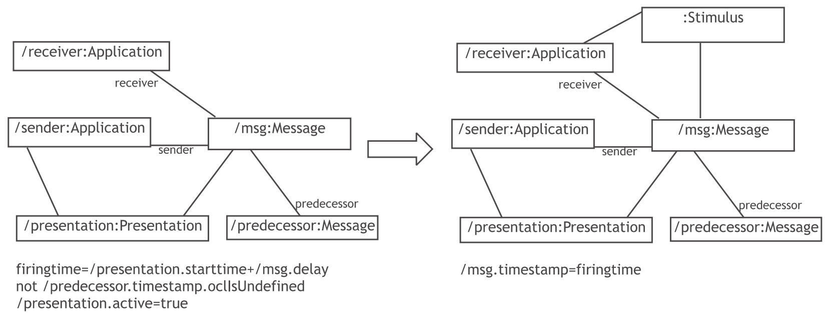


Fig. 10. DMM+t rule describing the sending of subsequent messages in an order

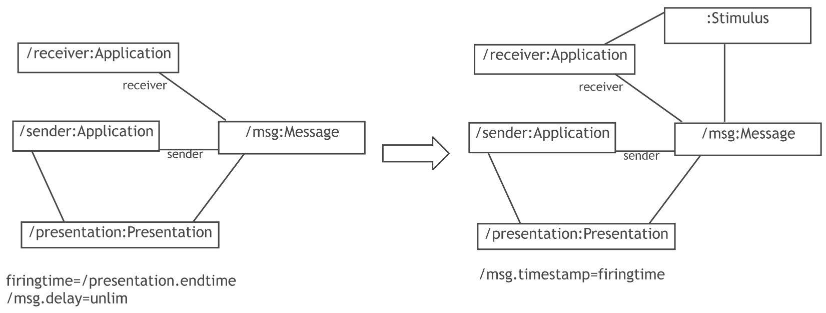


Fig. 11. DMM+t rule defining the sending of end-synchronized messages

straints. Only their preconditions differ, the effect (the end of the presentation) is the same for all three rules.

In contrast, the reception of a stopmessage requires the presence of a stimulus for the message as shown in Fig. 7. A stimulus is the UML instance of a message specification.[1] Whenever a message is sent, a stimulus is created and attached to the receiving object. In this way, sending and reception are decoupled. Thus one might e.g. formulate rules including protocol-based reception mechanisms (as specified by statecharts) as an alternative to direct reception. Since we focus on MM sequence diagrams, we provide direct reception rules for startmessages and stopmessages. The reception of a startmessage is specified in Fig. 8. The stimulus that indicates the pending message is consumed in the course of the rule application, and a device for rendering the media element is allocated. Note that as the firing time of both message reception rules is given as stimulus.chronos (the reception time of the stimulus), we assume a communication without observable delays. It would also be possible to model fixed delays, delays within a bounded interval or unbounded message delays by modifying this condition.

The third set of rules describes the sending of specified messages (Figs. 9 to 11). We have to distinguish between messages that have a numerically specified delay (either 0, indicating a synchronization with the start of the owning presentation, or a fixed time after that) and messages that have a delay value of unlim, indicating a synchronization with the end of the presentation. According to the UML specification, we also have to account for messages being in predecessor/successor relationship. The semantics specification results in three different rules describing the sending of specified messages.

[1] The precise definition of a Stimulus is not repeated here, it can be found in the UML specification [18, pp. 2–103ff].

Figure 9 displays the rule for the first message in a sequence that has a limited delay. Since the attribute msg.delay is present in the definition of the firing time, this could never be true for the value unlim (firing time can never be unlim). To interdict the application of the rule to any message which has a predecessor (and is thus not the first one), a negative application condition is used. A negative application condition (NAC) as defined in [11] contains a subgraph that must not be present in the context of a rule occurrence. It is indicated by enclosing the elements of the NAC in a dashed and canceled area. Therefore, this rule could never match the role /msg to any message in a given hostgraph that has a predecessor. The effects of the rule are the creation of a Stimulus at the receiving object and the specification of a timestamp for the message, thus indicating successful sending of the message.

The rule in Fig. 10 can be applied to any subsequent message in the order specified by the predecessor relationship. It requires the previous message to be executed (timestamp is set) and also requires a finite value for its delay. Note that if all messages would be required to carry a delay value, the way of processing messages could be simplified by determining the order based on their delay values. Combining the two ordering mechanisms can cause contradictions in the specifications (i.e., the specified order of messages and their delays do not correspond), but gives flexibility to combine timed specifications as used in our example with standard features of UML sequence diagrams. One could e.g. combine MM sequence diagrams with messages that are sent in reaction to some external event rather than after a fixed delay. Those messages would then have an unspecified delay, but could be placed in an order with the timed messages.

The remaining case that has to be specified leads to the resolution of an (intentional) ambiguity in the description of MM sequence diagrams in Sect. 3. The description in natural language does not yield any information on the situation that a presentation ends before all its messages are executed. The trivial options are either to send all remaining messages immediately (disregarding their delay values) or to discard them. When discarding the messages, we can furthermore distinguish whether end-synchronized messages should still be sent (disregarding the fact that some of their predecessors have been discarded) or whether they should also be discarded. Note that the two previous rules do not yield any information on this as they only apply in the case of active presentations. The rule in Fig. 11 does clarify these ambiguities. It states that at the end of a presentation the end-synchronized messages will still be sent. No condition requires the execution of previous messages, thus end-synchronization overrides the predecessor/successor order. There is furthermore no rule to process remaining messages that are not end-synchronized, thus they are discarded. This example shows how a formal specification strengthens and clarifies concepts presented in natural language although different semantic decisions could be taken.

4.3 Interpreting the semantics

One of the motivations for deploying formal semantics is the need to gain additional information on the model under consideration. Since specifications can be quite complex, it is not always obvious whether the specification has a meaningful interpretation or if it complies with the intention of the modeler. The definition of DMM+t rules facilitates an interpretation of a given model. The interpretation can be used for a visualization or a test of a given situation. This is especially important if certain elements of the model are underspecified, e.g. the duration of media elements is not given. Here, a modeler has to ensure that at least a few chosen test cases produce the intended behavior.

For such a test, an initial configuration (test case) has to be provided. In addition to the specification given in Fig. 3, we will assume the model to be in a state where all chronos values are initially 0, the timestamps are undefined, and all application objects are inactive. A trigger for the whole scenario is created by assuming that presentation p1 is to be started, i.e., p1.starttime is 0, p1 is active. We additionally assume a length of 50 seconds for Intro. A fragment of this configuration is depicted in Fig. 12.

The rule that can be applied on this initial configuration is the one shown in Fig. 9. The roles match as follows: /receiver on Tape Player, /sender on Projector 1, /presentation on p1, and /msg on m1. According to the OCL constraints of this rule the firing time will be at choronos=5 and a Stimulus will be created. Applying the rule of Fig. 8 will consume the stimulus and activate p2.

Abstracting from the actual details of these graph matchings, Fig. 13 shows an overview of the possible configurations and the transitions between them. In this figure, configurations are characterized by the chronos values of the four presentation objects and their application objects. Presentations that are currently active are shaded in grey. The names of the configurations are given as roman numerals. Labeled transitions between the configurations indicate the rules applied as well as the chronos value of the rule applications (the firing time). Rule applications with identical firing times have been combined into one transition.

The alternative paths between the configurations II and IV illustrate the effect of the global monotonicity theorem introduced in Sect. 2. Although some elements of configuration IIIa have already reached a chronos value of 200, performing operations on independent elements with lower chronos values (in the past) still remains possible. Whether an actual interpreter would compute the diagram by the path via configuration IIIa or configuration IIIb is non-deterministic. The theorem guarantees for each (successful) path of rule applications that an alternative path exists that is ordered with respect to the chronos values of the rule application (in the example the path via IIIb). Therefore, the result of the interpre-

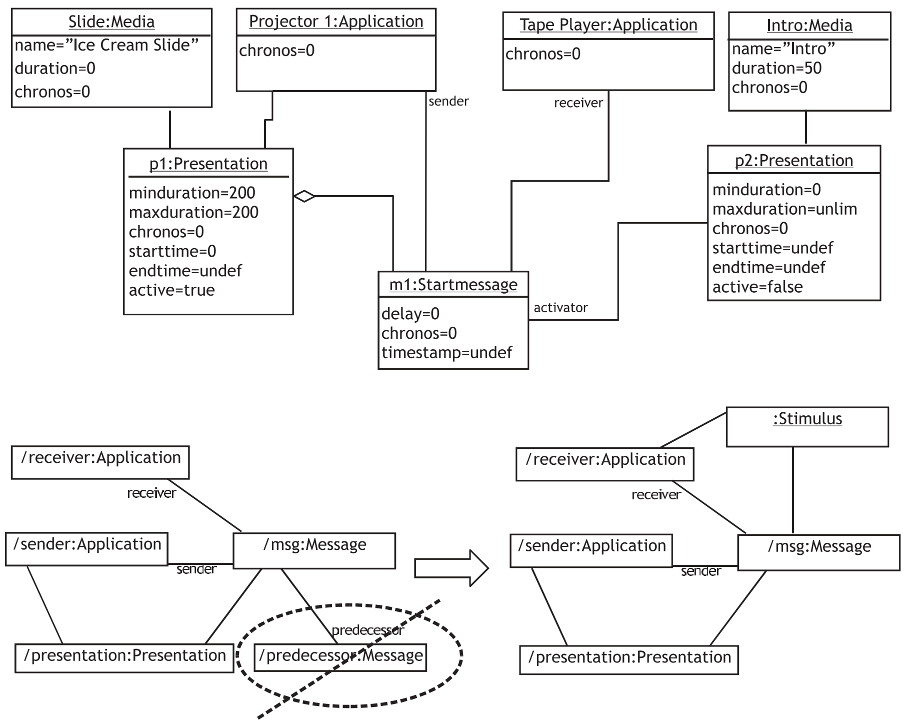


Fig. 12. Excerpt from the initial configuration

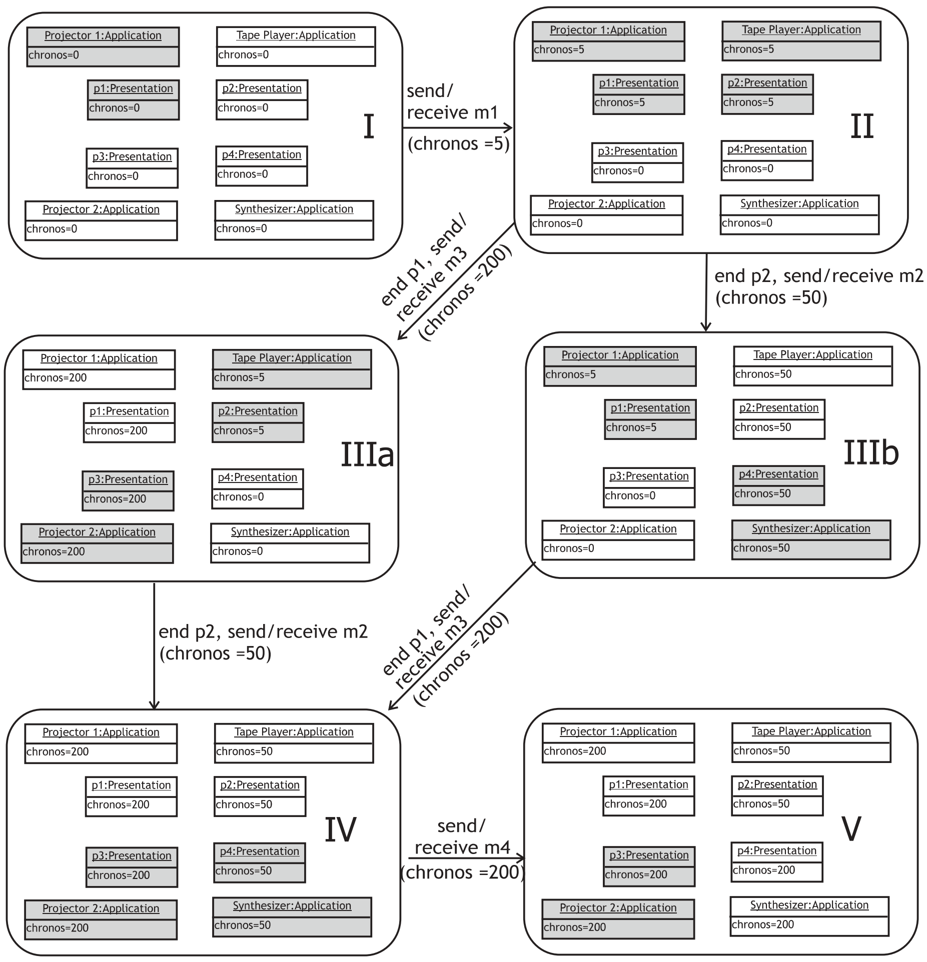


Fig. 13. Trace of the execution of the test scenario

tation is independent of the ordering of rule applications in an actual interpretation. The terminal configuration, i.e., the terminal instance graph, is not shown here; it is reached when the movie ends.

In general, a terminal state is reached when all presentation objects have been presented, that is, their endtime attributes are set. This intuitively corresponds to the completion of the scenario at the bottom of the sequence diagram.

More information on the application's behavior can be gained from possibly defective test cases, because the system's reaction to unexpected situations cannot easily be predicted. If we assume the length of the Intro to be 230 seconds, it is not intuitively clear how this situation is handled. If we apply our semantic rules to this scenario, we find that the scenario is invalid under the rules we specified. At the end of the slide presentation, a Stopmessage is sent to the tape player, but the resulting Stimulus cannot be consumed since the object is not active. Thus the Stimulus stays attached. At firingtime=235 the Intro ends and sends a Startmessage to the tape player. This can now be processed, the player starts. A conflict with the still lingering stimulus of the stopmessage does not occur as the conditions for processing the stopmessage are neither met before the startmessage is being processed (presentation not active) nor afterwards (chronos has been advanced). Thus the music never stops and no terminal state can be reached.

5 Strengthening the semantics

In the previous section we demonstrated how DMM+t rules can be used to specify semantics for UML diagrams and how these specifications can be put to use. Yet some details of the presented approach need to undergo a further investigation to guarantee that DMM+t rules are indeed able to express all concepts of MM sequence diagrams (and other dynamic UML diagrams with temporal aspects).

The first and most general observation is that building on graph transformations as the basis of the semantic domain gives a high degree of freedom. All graph transformation rules can be executed on any part of the graph provided their application conditions are satisfied. While this allows for a non-deterministic and concurrent execution of a specification, it has to be ensured that no unintended effects arise from this flexibility. The axioms given in Sect. 2 already restrict the handling of the chronos attributes to firing sequences that conform to the general concept of passing time. These axioms have been enforced by introducing the variable firingtime in the DMM+t rules and interpreting it accordingly. Thus the distributed presentation of media elements can be modeled using this semantic domain.

MM sequence diagrams provide some other notions that have to be properly represented. Outgoing messages from a presentation have an order (specified by their delays and/or the predecessor relation). To ensure that this order is preserved in the semantic domain, the rules for message processing are formulated to create a dependency between them. Each of the message-processing rules sets the timestamp of the message it executes and thus creates the context for the processing of the next message. Another intuitive order is embedded in the messaging mechanism. A message cannot be received before it is sent. Again, this is enforced by creating a dependency between the rules, based on the existence of the stimulus object.

But there still exists a degree of freedom in the semantic domain that yields non-intuitive results. Consider the MM sequence diagram given in Fig. 14 and assume that the media element assigned to presentation p1 has a duration of 80 (for the moment we will disregard p2 and p3 completely). If the presentation p1 starts at chronos=0, the rules for sending the message (at firingtime=100) and for reaching the end of the media object can both apply. If the message is sent, an awkward situation occurs. Not only does the sending of the message contradict the concepts specified in Sect. 3 and refined in Sect. 4. But also, by sending the message, the chronos value of the presentation object is advanced to a value of 100, thus the rule of Fig. 4 can never apply and end the presentation. Obviously, this is an unwanted behavior.

In terms of the formalization in Sect. 2 we have a conflict between two rules, which is unintended, because we want only the rule for ending the presentation to fire. There are two ways to rectify this situation. The first would be to introduce dependencies between all rules conflicting in this way. Those pairs of conflicting rules can be found e.g. employing the tool AGG [22]. This would result in a lot of artificially created elements embodying the dependencies. These would clog the specification, making it hard to understand and change, especially as each change might create new conflicts. Thus a more general solution has to be found.

The second and in this case much more elegant way to resolve this conflict is to introduce a general notion of priority. The human intuition for a resolution of this conflict would be that whichever rule would be "earlier", i.e., at a lower firing time, should be applied. We can compare the firing times since the rules are in

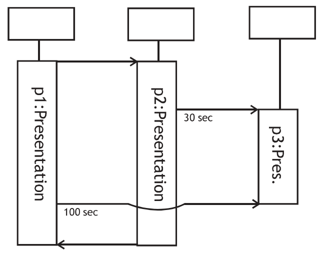


Fig. 14. Example of a complex MM sequence diagram

conflict and are thus influencing the local time of at least one common object (and all objects synchronized with it by the rule). Thus we state that an occurrence of a rule may only happen if there is no conflicting rule that might occur at a lower firing time. We call the semantics conforming to this restriction the *locally strong semantics*. By requiring this kind of priority it is guaranteed that the local order of rule occurrences does indeed conform to the intuitively expected order of events.

This order is still enforced only locally. It does not prevent "unintuitive" firing sequences due to the lack of synchronization of local clocks. Refer again to Fig. 14. If we assume the media of p1 to have a duration of 80 and p2 to have a duration of 70, we might expect that the end-synchronized stopmessage sent by p2 will stop p1 before it reaches its normal end. But once the presentations p1 and p2 are started, both the rule for ending p1 due to the end of the media (it has a lower firing time than the conflicting rule for sending the message at firingtime=100) and the rule for sending the message from p2 to p3 at firingtime=30 can occur (it is not in conflict with the rule for ending p1 as there are no common elements). Thus, non-deterministically, we might end p1 and thereby advance its chronos to 80. Then at firingtime=70, p2 would end, but the rule for emitting the stopmessage to p1 would not be applicable since the chronos value of p1 would already be advanced to 80. Note that this is not a faulty behavior if we regard the different media objects as unsynchronized entities with their own local clocks. Under this assumption, the sequence described above could yield valuable information for the case of a very "slow" componen rendering p2.

But not every timed UML diagram assumes this kind of local clocks. In most cases it is much simpler (and justifiable from the appliation domain) to assume perfectly synchronized components, i.e., a global clock. In this case, yet another restriction can be placed upon the semantics: we can require an interpreter to always choose from all possible rule occurrences one with minimal firing time. This leads to the automatic creation of a globally time-ordered firing sequence in which no object can "overtake" another. In [8] this concept is called the *strong semantics*. This kind of semantics places a heavy restriction on the non-deterministic nature of the underlying formalism, as it only allows for a non-deterministic choice between possible rule occurrences that happen at the same firing time. This interpretation of the rules is closest to the intuitive human interpretation.

We imagine that a tool could provide both possible intrepretation mechanisms. In that way, a modeler could first interpret his specification under the assumption of perfect time and then move to an interpretation taking distributed clocks into account (if this is required by the application domain). We believe that both interpretations yield interesting new and valuable information on the model.

6 Conclusion

In this paper, we have extended the approach of Dynamic Meta Modeling [2, 9] to specify the semantics of time-dependent dynamic behavior of UML models. As a case study, we have applied this approach to multimedia sequence diagrams, a variant of UML sequence diagrams for modeling the control of multimedia presentations.

We have taken a high-level point of view in two different respects. First, the proposed semantics is at the requirements level, that is, we have assumed zero duration for the operations like the start or termination of a presentation or the transmission of a message. Furthermore, we did not consider failures and delays due to imperfect infrastructure (e.g. lack of resource availability), that may appear for instance in distributed Web-based multimedia applications. To capture those aspects, an extension of both the language of multimedia sequence diagrams and its semantic rules is required.

Second, the rules themselves are high-level because they assume an execution mechanism based on global pattern (that is, graph) matching which provides non-determinism or backtracking to search for a successfully terminating sequence. Although there are tools supporting these paradigms [21], this is quite different from standard object-oriented concepts. It is a topic of future research how to map abstract semantic rules to object-oriented implementations.

References

1. David A, Möller MO, Yi W (2002) Formal verification of UML statecharts with real-time extensions. In: Proc. 5th International Conference on Fundamental Approaches to Software Engineering (FASE 2002), Lecture Notes in Computer Science, vol 2306. http://www.springer.de/comp/lncs. Springer, pp 218–232
2. Engels G, Hausmann JH, Heckel R, Sauer S (2000) Dynamic Meta Modeling: A graphical approach to the operational semantics of behavioral diagrams in UML. In: Evans A, Kent S, Selic B (eds) Proc. UML 2000, York, UK, Lecture Notes in Computer Science, vol 1939. http://www.springer.de/comp/lncs. Springer-Verlag, pp 323–337
3. Ehrig H, Pfender M, Schneider HJ (1973) Graph grammars: An algebraic approach. In: 14th Annual IEEE Symposium on Switching and Automata Theory. IEEE, pp 167–180
4. Engels G, Sauer S (2002) Object-oriented modeling of multimedia applications. In: Chang SK (ed) Handbook of Software Engineering and Knowledge Engineering, vol 2. World Scientific, Singapore, pp 21–52
5. Firley T, Huhn M, Diethers K, Gehrke T, Goltz U (1999) Timed sequence diagrams and tool-based analysis – A case study. In: France R, Rumpe B (eds) Proc. UML '99, Lecture Notes in Computer Science, vol 1723. Springer-Verlag, pp 645–660
6. Gyapay S, Heckel R, Varro D (2002) Graph transformation with time: Causality and logical clocks. In: Corradini A, Ehrig H, Kreowski H-J, Rozenberg G (eds) Proc. 1st International Conference on Graph Transformation (ICGT 02), Barcelona, Spain, Lecture Notes in Computer Science, vol 2505. Springer-Verlag, pp 120–134
7. Gyapay S, Heckel R, Varro D (2003) Graph transformation with time In: Fundamenta Informaticae 58(1):1–22

8. Ghezzi C, Mandrioli D, Morasca, Pezzè, S (1991) A unified high-level Petri net formalism for time-critical systems. IEEE Transactions on Software Engineering 17(2):160–172
9. Hausmann JH, Heckel R, Sauer S (2001) Towards dynamic meta modeling of UML extensions: An extensible semantics for UML sequence diagrams. In: Proc. IEEE Symposia on Human-Centric Computing Languages and Environments (HCC'01), pp 80–87
10. Hausmann JH, Heckel R, Sauer S (2002) Dynamic Meta Modeling with time: Specifying the semantics of multimedia sequence diagrams. In: Bottoni P, Minas M (eds) Proc. International Workshop on Graph Transformation and Visual Modeling Techniques (GTVMT 2002), Barcelona, Spain, Electronic Notes in Theoretical Computer Science 72(3). `http://www.elsevier.nl/locate/entcs`. Elsevier Science
11. Heckel R, Wagner A (1995) Ensuring consistency of conditional graph grammars – A constructive approach. In: Proc. of SEGRAGRA'95 "Graph Rewriting and Computation", Electronic Notes in Theoretical Computer Science, vol 2. `http://www.elsevier.nl/locate/entcs`. Elsevier Science
12. Kreowski H-J (1977) Manipulation von Graphmanipulationen. PhD thesis, Technical University of Berlin, Department of Computer Science
13. Küster JM, Stroop J (2001) Consistent design of embedded real-time systems with UML-RT. In: Proc. IEEE Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2001), pp 31–40
14. Li X, Lilius J (1999) Timing analysis of UML sequence diagrams. In: France R, Rumpe B (eds) Proc. UML '99, Lecture Notes in Computer Science 1723. Springer, pp 661–674
15. Guldstrand Larsen K, Pettersson P, Yi W (1997) UPPAAL in a nutshell. International Journal on Software Tools for Technology Transfer 1(1–2):134–152
16. Mühlhäuser M, Gecsei J (1996) Services, frameworks, paradigms for distributed multimedia applications. IEEE Multimedia 3(3):48–61
17. Object Management Group (2002) UML Profile for Schedulability, Performance, and Time, OMG adopted specification
18. Object Management Group (2003) OMG Unified Modeling Language Specification, version 1.5
19. Petriu DC, Shen H (2002) Applying the UML performance profile: Graph grammar-based derivation of LQN models from UML specifications. In: Computer Performance Evaluation, Modelling Techniques and Tools (Proceedings of TOOLS 2002), Lecture Notes in Computer Science, vol 2324. Springer-Verlag, pp 159–177
20. Sauer S, Engels G (2001) UML-based behavior specification of interactive multimedia applications. In: Proc. IEEE Symposia on Human-Centric Computing Languages and Environments (HCC'01), pp 248–255
21. Schürr A, Winter AJ, Zündorf A (1997) PROGRES: Language and environment. In: Ehrig H, Engels G, Kreowski H-J, Rozenberg G (eds) Handbook on Graph Grammars and Computing by Graph Transformation: Applications, Languages, and Tools. World Scientific, Singapore, pp 487–550
22. Taentzer G (1999) AGG: A tool environment for algebraic graph transformation. In: Proc. International Workshop on Applications of Graph Transformations with Industrial Relevance (AGTIVE 1999), pp 481–488

Jan Hendrik Hausmann is a PhD student at the University of Paderborn, Germany. His research topics include object-oriented modelling with the UML, graph transformations and their application in software engineering, meta modelling, and eLearning systems.

Reiko Heckel is assistant professor at the University of Paderborn, Germany, since 1998. His research interest is the use of graph transformation in software engineering, including the development of relevant theory like structuring and modularity concepts, concurrency theory, graph-based temporal logic, etc. and the application of this theory to the modelling of object-oriented and agent-based systems, and to the semantics of visual modelling languages.

Stefan Sauer works as a research and teaching associate at the University of Paderborn in the database and information systems group. The focus of his research is on object-oriented modeling with the UML, semantics of visual languages, meta modeling, extensions of the UML for interactive multimedia applications, and multimedia software engineering.