Identifying Semantic Dimensions of (UML) Sequence Diagrams

Jan Hendrik Hausmann, Jochen Malte Küster, and Stefan Sauer

University of Paderborn
Department of Mathematics and Computer Science
D-33095 Paderborn, Germany
{corvette|jkuester|sauer}@uni-paderborn.de

Abstract: Although UML sequence diagrams are widely used in practical software development, there is still a great demand for improvements. Their use both within and outside the standard interpretation of the UML specification is not seldom confused because different interpretations for sequence diagrams exist without means to distinguish between them. Furthermore, alternative sequence diagram notations with more syntactical features and different semantics still have a big influence and are readily used (explicitly as well as implicitly) alongside UML's sequence diagrams. Without necessary clarification, the meaning of sequence diagrams remains vague. Hence they are not suited for providing a common understanding of inter-object behavior which is a prerequisite for their deployment within rigorous software development processes. Additionally, model quality assurance by consistency checking and validation is not well supported. In this paper, we survey, structure, and classify syntactic and semantic alternatives that appear in sequence diagrams in practice. We thereby identify scope of interpretation, level of abstraction, composition and refinement, ordering, time, and represented function as the essential semantic dimensions of sequence diagrams. The spanned semantic space is suited as a basis for discussing and proposing extensions of UML sequence diagrams to precisely determine the semantic interpretation of modeled sequence diagrams.

Keywords: UML sequence diagrams, semantic space, semantic variation points, semantic dimensions, extensions, syntactic transformations

1 Motivation

UML sequence diagrams [OMG01] are one of the most widely used UML behavioral diagrams in practical software engineering. Their very intuitive presentation of message sequences along a time axis qualifies them as the diagram of choice for depicting scenarios or protocols of behavior. According to the UML specification, interaction diagrams (i.e., sequence and collaboration diagrams) are intended to describe the realization of operations or classifiers (e.g. use cases). In general, their use is not restricted to certain communities or methodologies, but is rather common among the users of the UML. Although this widespread use would indicate that the UML community has reached (at least partially) a common understanding of what is expressed by a sequence diagram, this does indeed not seem to be the case. When surveying scientific publications [Oe99,Kn99,HHS01], development methodologies [BJR99,WD99] and software development documentations, numerous interpretations of UML sequence diagrams can be found. Furthermore, the influence of alternative notations (MSCs [ITU96], LSCs [DH98]) still seems to be very strong, so that UML sequence diagrams face a heavy ambition to use the (seemingly) more powerful or more adequate notations and concepts of these alternative diagrams (see e.g. the responses of Verilog, Alcatel, Ericsson and Motorola to the UML 2.0 RFI [OMG99]). We do thus face a situation in which many people take a great interest in sequence diagrams, but their scope and interpretation in the context of the UML is rather unclear. But not only the practical usage of sequence diagrams lacks precision, defined software development processes seem to ignore the problem as well. Section 3 gives two interpretations of sequence diagrams, both drawn from the Rational Unified Process [BJR99], that have strong semantic differences, but are not clearly distinguished in the process description. This situation will lead to disoriented software developers at best and severe misdevelopments at worst. To improve this, a clear understanding of the semantic alternatives of sequence diagrams has to be developed. It has to be examined which of these alternatives are supported by UML sequence diagrams and how to state which interpretation should underlie a given sequence diagram.

In this paper, it is our intention to identify the most common ways in which sequence diagrams (UML and other) are used. We will separate syntactic from semantic issues and group semantic alternatives along orthogonal dimensions, thus creating what we call a *semantic space* for sequence diagrams. Each point in this space will represent a precise interpretation for a sequence diagram. By establishing a mapping from syntactic elements of sequence diagrams to these points it will become possible to provide each sequence diagram in a development process with unambiguous semantics. Precise and unambiguous semantics of a diagram type is a mandatory prerequisite for its use within rigorous software development processes. Note that it is not an aim of the paper to assess or to compare different approaches but rather to survey, sort, and structure the possibilities implied by the usage of sequence diagrams. The organization of the paper is as follows: Section 2 introduces the criteria and methodologies underlying the following sections. Sections 3 to 8 each investigate a certain dimension of the semantic concepts of sequence diagrams, thereby creating the semantic space. The concluding sections 9 and 10 summarize the results of the survey and introduce ideas for their usage.

2 Criteria Underlying the Survey

Syntax and semantic concepts of every language should be tightly coupled. Having a semantic concept without a construct of the abstract syntax to express it is as useless as having syntactic elements (in the concrete or abstract syntax) that do not carry any meaning. The UML is no exception to this rule, but does rather introduce a third problem of having multiple possible interpretations without syntactic symbols to choose between them. What we will conduct in the following sections is an investigation which notations and interpretations for sequence diagrams are commonly used in practice or research. We analyze whether they introduce new semantic ideas (as opposed to syntactic abbreviations) and which syntactic elements may be used to identify the semantics. The distinction between syntactic abbreviations and semantic extensions can be made by examining if two different sets of syntactic elements cover the same semantic concepts.

We will furthermore make a distinction between semantic choices that are mutually exclusive (e.g. specification on the role or the instance level) and semantic choices that are orthogonal (e.g. having time in a model and specification on the role level). Ordering all mutually exclusive choices along a dimension, we get a semantic space which contains all possible semantics of sequence diagrams. In terms of the UML, the choices along one dimension represent a *semantic variation point.*

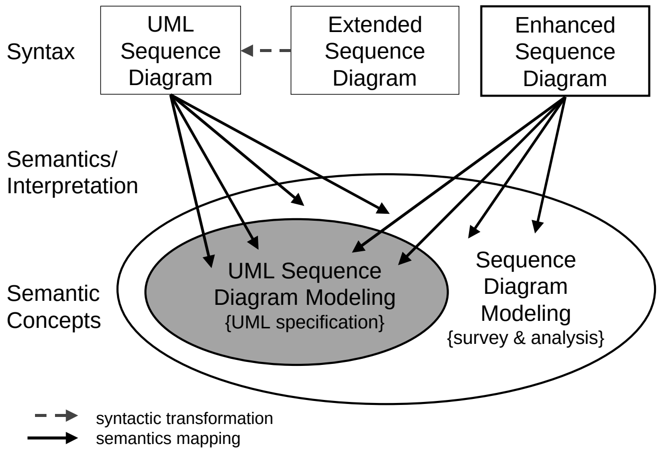
Syntax
UML Sequence Diagram
Extended Sequence Diagram
Enhanced Sequence Diagram
Semantics/ Interpretation
Semantic Concepts
UML Sequence Diagram Modeling {UML specification}
Sequence Diagram Modeling {survey & analysis}
syntactic transformation
semantics mapping

Fig. 1: The semantic space of sequence diagrams

As we do not strictly focus our survey on UML sequence diagrams, the semantic space produced will include several semantic concepts that are not supported by the current UML specification. Thus, we have to identify the subspace containing exactly the UML semantics of sequence diagrams. Figure 1 illustrates the basic notions used in this paper: Based upon the standard syntax of the UML, different semantics may be identified, some inside, some outside the scope of UML semantics. Furthermore, syntactically *extended* sequence diagrams may either be semantically equivalent to UML sequence diagrams (purely syntactic extensions) or introduce new semantic ideas (syntactically and semantically *enhanced* sequence diagrams).

3 Scope: Specification or Scenario—Mandatory vs. Possible Behavior

In order to gain an understanding of the possible circumstances of use for UML sequence diagrams, it is necessary to first obtain a high-level view of their capabilities. UML sequence diagrams—as well as their predecessors and alternatives [DH98,ITU96]—may have different meanings in the scope of an overall model. On the one hand they are employed to illustrate an especially interesting interaction involving multiple objects. By using sequence diagrams this way it is possible to capture the expected behavior as well as exceptional situations. We call this way of usage *scenarios*. Semantically, a scenario is a requirement for the behavior of the modeled system. The term scenario and the loose interpretation of this kind of sequence diagrams are mainly used in the area of requirements engineering. If no further restrictions are made on the context of the scenario, it only expresses that the shown sequence of messages should be *possible* in the final, implemented system. Additional messages and objects may be needed to produce the intended behavior. Thus, in its weakest form, a scenario only requires that a system that is built according to the model may produce a trace that contains the specified messages in the specified order. In this very loose interpretation, a sequence diagram is of limited use as a modeler certainly expects more than a single possible trace of the system to fulfill a modeled scenario.

Another way to use sequence diagrams is to exactly capture the intended behavior of a system. In this interpretation, any system producing a trace that differs from the specified sequence diagram would be inconsistent with the model. We call this interpretation of sequence diagrams *specification*. As a sequence diagram does usually not cover all possible interactions of a system, the specification-type diagrams are commonly extended by a precondition that implies the occurrence of the specified interaction. An example for this kind of usage is given in [BJR99] where test cases are expressed in terms of sequence diagrams. It is also possible to weaken the semantics of such a diagram by checking only *projections* of the traces against the sequence diagram. If only certain parts of the sequence diagram must have a meaning of being *mandatory*, while others are *optional*, projection conditions on the specifying sequence diagram can be used to ensure a match of the diagram and a program trace. By using projection conditions the strict semantics of specification-type sequence diagrams can be loosened to enable more abstract modeling. What is not covered by the semantic concepts of the standard UML is the idea of *anti-scenarios*. These are used to specify bad examples, i.e., interactions that are considered invalid. Like specifications, anti-scenarios affect all possible traces of an implemented system. Therefore, projection conditions can be applied to anti-scenarios as well.

The dimension of *scope* is thus organized along the strictness of the diagram's interpretation. It starts with the loose semantics of the scenario, contains interpretations that are strict under certain projection conditions and ends with the strict semantics of the specification.

In the UML specification, scenarios and specifications are mentioned as possible usages for a sequence diagram, but the semantic differences are never investigated nor are explicit syntactic constructs given to distinguish between them. As a modeling process typically moves from the more abstract and loosely coupled diagrams towards a tightly coupled system model, it would be profitable to define a set of sequence diagrams with differing levels of strictness. These are to be used in the different stages of the modeling process.

4 Abstraction: Instance Level vs. Role Level

The UML specification explicitly introduces two different kinds of collaborations (the construct underlying sequence diagrams and collaboration diagrams). On the one hand, the participants of the collaboration may be actual objects. This can for example be useful when employing sequence diagrams to visualize program traces [MW01], since roles of runtime objects are generally not derivable from the program code. Also for program testing, it is often necessary to distinguish between different particular instances (possibly conforming to the same role). On the other hand, sequence diagrams may represent a communication pattern between roles. A sequence diagram interpreted in this way defines an interaction between objects that fulfill certain criteria to play the specified roles. Thus it abstracts from particular objects or allows to focus on distinguished properties that make up a particular role in the given context. For instance, this way of usage is profitable when employing design patterns to develop a software system. The prototypical interaction of the pattern can be displayed in the form of a sequence diagram and can be deployed at several locations throughout the software system only by referencing the pattern as a whole. Both of these semantic variants are exhaustively explained in the UML specification. The syntactic distinction between the two types of diagrams is made via the participants involved in the interaction. If these participants are marked as objects (underlined name), a diagram on the instance level is intended. If the participants do not have underlined names, we get a role-level diagram (specification-level in UML terminology, but we distinguish specification as a point of the scope dimension; see previous section). If both kinds of participants occur in the same diagram, this is considered incorrect. It may be argued that a mixture of roles and instances may have certain benefits, but we are not aware of an approach that actually promotes such a kind of sequence diagram. We do thus believe that the dimension regarding the level of *abstraction* only consists of two points, the instance and the role level.

5 Composition and Refinement

The notational elements of UML sequence diagrams are rather satisfying for the modeling of simple scenarios and procedural message flow. But when it comes to the modeling of more complex patterns or instances of interaction, it may be advantageous to have additional language features to structure and modularize sequence diagrams. These are control flow operators such as iteration or conditional branching on the one hand, and composition (and decomposition) operators on the other hand.

The UML specification introduces branching and iteration within sequence diagrams only as presentation options for messages and stimuli ([OMG01], section 3.63.3). Regarding modularization, another presentation option ([OMG01], section 3.61.3) allows a modeler to link sequence diagrams by dangling message arrows leaving (entering) a sequence diagram. The counterpart sequence diagram where the message continues (from which the message originates) is only textually referenced by name using a note. Semantically, the linked diagrams are parts of one underlying interaction. These are of course rather restricted forms of structuring.

Branching is a purely syntactic extension of sequence diagrams that may be factored out into alternative interaction flows with equivalent semantics. In addition to multiple diagrams depicting particular occurrences of the possible execution paths, an interaction can thus alternatively be described using a single diagram including conditional flow of communication. Iteration can be viewed as an extension of sequence diagram on both the syntactic and the semantic level. If iterations only represent finite loops, they are a purely syntactic extension, although a very convenient one in practice. If loops may be infinite, then iteration becomes a semantic enhancement that can not be expressed by rolling out the iteration. We do not further investigate this case here.

Composition can be viewed from different perspectives. The first alternative is composition of behavior for single objects. This means that the reaction of objects when receiving a message, i.e., the activations of objects (see also Sect. 8), may be decomposed and refined into concurrent or sequential parts in order to show the encapsulation of more complex behavior (e.g. used in [SE99]). The second alternative is refinement on the structural level, i.e., replacing an object on a sequence diagram by a set of collaborating objects ([OMG01], section 3.63.3). This automatically implies the decomposition of behavior as well: Activations of the original object are decomposed into activations of the collaborating objects and interactions (i.e., messages) between these objects causing their activations.

For example, in the field of multimedia applications it is often emphasized that the reaction to a media presentation invocation is a complex process of its own, containing prefetching, preprocessing, presentation, and postprocessing activities. Any of these activities can be subject to temporal constraints. This complex behavior may be encapsulated by a single message (activation) on a sequence diagram which may then be further refined if necessary in another step of a multimedia development process (compare [HHS01]).

While the previous concepts represent local refinement, more general forms of composition may operate on the granularity of sequence diagrams. Sequence diagrams may be sequentially or concurrently composed, or they may be refined by nesting sequence diagrams representing complete patterns or instances of interaction within higher-level sequence diagrams.

The concept of refinement implies the need for consistency between more abstract and more detailed parts of the model. When we conceive refinement as the integration of more detailed parts of a model in a more abstract model part, then we need to define what the interface between these two models has to look like.

Composition and refinement operators can be discussed on the syntactic and the semantic level. They can be regarded as purely syntactic extensions that can be flattened to simple existing syntax elements and, consequently, are mapped to traditional semantics. But they can also be seen as enhancements of sequence diagrams on the semantic level. In the latter case, it needs to be assured that the operators on both the syntactic and semantic levels and the mapping of syntactic elements to the semantic concepts are commutative.

6 Ordering of Events

Although the ordering of messages in a sequence diagram reflects causal dependencies, sequence diagrams depict the exchange of messages (as well as stimuli) in a temporal order. One can distinguish between a partial and a total order. Another distinction arises from the elements that are ordered. These can either be the messages themselves or send and receive actions of the messages. In the following, we will denote such actions also as events as this is common in other sequence diagram modeling notations.

Concerning UML sequence diagrams, the UML meta model states that there exists a partial order of messages as follows: Messages are ordered by predecessor and activator relationships. (While activator relations represent control-flow dependencies that are explicitly causal, predecessor relations may be implicitly causal, e.g. based on data dependencies, or just an arbitrary sequential ordering undertaken by the modeler.) This partial order well reflects the possible order of messages in collaboration diagrams modeled with sequence numbers. As in collaboration diagrams, sequence numbers can be used in sequence diagrams to explicitly express (procedural) nesting, sequencing, and concurrent threads of communication including synchronization. However, sequence numbers are often omitted in sequence diagrams since the graphical position of message arrows (i.e., visual information) shows relative sequences. Nevertheless, these alternative (syntactic) representations must be consistent if they are used together, or the necessary ordering information needs to be derived from the graphical diagram notation. This leads to the problem of consistently mapping graphical relations of message arrows to the activator and predecessor meta model relations. If sequence diagrams increase in complexity and concurrency, ambiguity of the graphical notation may arise. Especially regarding predecessor relations within concurrent sequence diagrams, both the sequence relative to which successive messages are ordered (i.e., the scope of the graphically implicit order) and possibly multiple concurrent preceding messages must be identified to obtain unambiguous semantics.

Another problem is that the actions of sending a message and receiving a message, that can be visually distinguished in concrete syntax if transmission time is not negligible, are not distinguished on the level of abstract syntax. As a consequence, overtaking of messages can only be indirectly expressed in UML interactions by incorporating time constraints (see next section). However, this kind of relations is part of the ordering dimension.

A possible form of partial ordering consists in having all events on a lifeline of an object ordered. In addition, all events related via a *send* and *receive* actions of a message are ordered. Thus, from the viewpoint of an event, it can only be assured that all following events on the lifeline of the same object will occur at some later point in time. Moreover, all receive events from outgoing messages of this event will be received after their emission. But it can for example not be assured that the reception of an outgoing message on some remote object occurs before the sender object commences with the next operation (in case of asynchronous messages).

One justification for employing partial orders within a sequence diagram is to leave aspects not relevant to the model unspecified. For example, if a sequence diagram depicts the message exchange of concurrently executed objects, it might not be necessary to determine a total order. It may be feasible to introduce even weaker partial orderings by allowing certain events on the timeline of an object to switch places, thereby enabling a concurrent modeling of a set of events, in which no order is fixed. This would be in accordance with the idea of concurrent co-regions as defined in MSCs [ITU96].

On the other hand, there are also applications of sequence diagrams where a total order might be required. For example, when using sequence diagrams for deadlock detection and for displaying the trace of a deadlocked program (see [MW01]), it might be necessary that exactly one path of message exchanges is fixed by totally ordering all messages and their receive/send actions. Older versions of the UML supported this idea by supplying a general timeline on which every event had to map via its vertical position in the diagram and which thus produced a total (temporal) ordering. In the current version of the UML, this can only be achieved by applying additional constraints, but it is not expressible using graphical notation (unless the graphical ordering is globally interpreted, i.e., predecessor relations are globally derived in the scope of the overall diagram for all messages).

Such additional constraints can be used for incrementally transforming a partial order into a total one. Unlike a total ordering based on some intuitive model of total ordering (as arranging the events along a vertical timeline), the specification of a set of constraints bears the risk of contradictions; consistency checking has to take place. Regarding the proposal to express program traces in UML sequence diagrams, the specification of a lot of constraints for a quite simple intuition appears to be unpractical. It seems to be desirable to have the option to fix a total order by specifying a corresponding semantics that interprets visual information. This ensures that the ordering of two events is determined by their relative vertical position in the diagram. This semantic alternative is currently not supported by the UML semantics.

A comparison of these alternative approaches reveals the following insights: A sequence diagram is either partially ordered or it is totally ordered. Concerning the partial order, different degrees of such an order are possible, one of them being the one currently found in UML. A total order may either be produced by adding enough constraints to a partial order or by interpreting the position of the elements in a fashion that leads to a total order. The dimension of ordering therefore distinguishes partial ordering (that may be further refined) and total ordering.

7 Time Quantification

As a sequence diagram depicts the exchange of messages in a temporal order, the question arises which model of time is underlying the sequence diagram and which time concepts are supported. It is obvious that the time dimension is strongly related to the ordering dimension discussed in Sect. 6. In their most primitive form, the temporal relation between two events (as introduced in the previous section) is not quantified. In this case, the time model is only qualitative and collapses to the concept of (temporal) ordering. In contrast, the temporal relationships may be quantified by deploying some time metric that relates events more precisely in time. This is important for the specification of timing requirements as they are mandatory in time-dependent applications such as real-time or multimedia systems. Obviously, one can further distinguish between discrete and continuous time models, but we will not focus on this difference here since their respective implications for sequence diagrams must be discussed in detail first to make such a distinction useful for sequence diagram modeling.

Another distinction is that of time models based on points in time or time intervals. Since interval-based representations can be mapped to point-based representations, this distinction does not have semantic implications that are important for our discussion. Nevertheless, both alternatives should be intuitively supported on the syntax level.

What can be further distinguished is the aspect of uniformity of the temporal metric along a sequence diagram. This refers to the idea of conceptual mismatch that is often discussed in the area of visual languages. It denotes the discrepancy of the human intuition and perception and the notation of the visual language. For example, when two activations are depicted one with a short, the other with a long activation symbol, then the intuitive interpretation is that the activity represented by these symbols have a temporal duration that coincides with the graphical length of the symbols. This means that a user implicitly assumes a sequence diagram to be a kind of temporal grid representation that has a uniform time base (or unit), resulting in a linear time axis. Temporal relations can be derived from the absolute and relative positions of events according to the specified time unit, i.e., the graphical symbols of sequence diagrams are interpreted as visually modeled temporal requirements or constraints (this already holds for the ordering dimension discussed in the previous section). Another semantic question in this context is whether a notion of time exists before the first and after the last message of the depicted interaction or if an object has a notion of (local) time when it is inactivated (e.g. supported by the concept of an external timer).

In contrast to an implicit specification of time by a linear, metric time axis, it is also possible to explicitly state temporal quantifications on sequence diagrams. This alternative allows a modeler to scale models or to vary the unit of time measurement and to model non-uniform progress of the time represented on a time axis relative to a global, perceivable (real) time. By using explicit temporal notions, absolute and relative timing information can be modeled and temporal requirements and constraints can be stated more precisely. Especially since the graphic notation is limited with respect to the expressiveness of temporal properties, textual constraint notations may be used as a syntactic variant. For example, if a temporal instance is defined relative to another temporal instance that can not be graphically identified (because there are alternative events that are temporally identical in the graphical notation, e.g. send and receive times for horizontally drawn message arrows or multiple arrows originating from the same vertical level) then it is ambiguous to which it refers, possibly causing semantic inconsistencies or update anomalies, e.g. if the reference instance is shifted.

An (explicit) time constraint is a constraint formulated over the time of two or more events. In UML, time constraints are Boolean expressions containing time expressions. They can express absolute timing or timing relations relative to other temporal instances. Time expressions are formulated using time functions on stimuli and message names ([OMG01], sections 3.60.3, 3.64). Constraints are defined in the scope of the entire diagram. Sequence diagrams support time constraints by the use of the OCL or other constraint languages, or graphically by construction marks for time intervals as a presentation option. Additionally, approaches exist for verifying the consistency of such timing constraints.

Sometimes a global time axis and local time axes for each object are distinguished. These two different interpretations both have their justifications in systems that are modeled with sequence diagrams. In concurrent systems, without applying additional techniques, we face a system with local clocks. As a consequence, in the first place we can only make assumptions on time occurring on one lifeline. On the other hand, when modeling a one-processor system where a number of objects are executed, it is straightforward to assume a global clock and thus to use a global time axis. Also, in concurrent systems, global time might be established using specific synchronization techniques and therefore there might be usages of sequence diagrams for concurrent systems with a global time axis.

The concept of global and local time axes refers to a combination of time and ordering (see Sect. 6) in the sense that events on a local time axis are only temporally related to other events on the same time axis, while a global time axis implies that all events are mapped to a common time axis. Therefore, although they are called time axes, they represent rather the concept of ordering that may (or may not) be quantified by time metrics. But even in the case of a global time axis, it is still possible to either require all events to be ordered relative to it or to allow events to be unspecified regarding their temporal ordering. This means that the global ordering may be partial, alike events on the local time axis of a single object may not all be ordered. With respect to timing, a time unit can be assigned as the time base to a single or multiple time axes, or even to segments of axes.

Semantically, we distinguish temporal ordering, implicit and explicit timing as the alternatives for the timing dimension in the following.

8 Function View: Interaction vs. Internal Activity

The traditional use of (UML) sequence diagrams focuses on the presentation of interaction between objects by exchanging messages to invoke operations, send signals, create or destroy instances. In this intention, the reception of a message causes the presence of an activation symbol—denoting focus of control—on the lifeline of the receiver object (at least for passive objects) to explicitly represent activity, e.g. method execution. However, the reaction of the receiver object is limited to succeeding messages sent by this object. Although an activation symbol appears within the concrete syntax of UML sequence diagrams (i.e., as a diagram element), there is no direct representative for it on the level of abstract syntax (i.e., a model element), given by the UML meta model.

Surprisingly, the UML semantics description (in the notation guide, [OMG01], section 3.62.1) gives an interpretation for this purely notational detail. As this semantics is quite intuitive, many people use it. The consequence of this situation is that all properties of an activation must formally refer to received or sent messages. Hence any approach to use activations for more than an intuitive annotation must fail.

According to the UML specification, an activation denotes the period of time during which an object (directly or indirectly) performs an action. It comprises the control relationship to its caller as well as initiation and completion times delimiting its duration. In procedural flow of control, this timing information can be strictly bound to incoming and outgoing, (possibly omitted) return messages. This might not always be the case for concurrent flow of control.

The performed action may only be given as a textual label or be indicated by the incoming message. For procedural flow of control, it denotes the interval during which a procedure or a subordinate procedure is active. For concurrently active objects with their own threads of control, an activation represents the performing of an operation or a transition in a state machine ([OMG01], section 3.62.2), independent from operations of other objects. Thus activations support more behavioral expressiveness than just exchanging messages between objects.

In addition to observable behavior in the sense of succeeding messages, sometimes it may be intended to also specify some part of the internal reaction of the receiver object, e.g. internal computations, actions or activities, in a sequence diagram.

The lack of a model element for activations becomes especially critical in the domain of (distributed) real-time systems. In this domain, it may be necessary to specify timing constraints on the duration of internal actions or computations, and not just between different events of observable messages. Additionally, the duration of an (asynchronously invoked) activation that does not terminate by sending or receiving a message, cannot be semantically modeled since timing constraints can only be formulated using timing functions on message and stimuli names (see Sect. 7).

An activation in a sequence diagram is supposed to show where the corresponding object is active. The focus of control is passed to another object when making a synchronous method call. There might be different types of activations depending on the usage of sequence diagrams. In an analysis diagram, the activation may only be needed for keeping track of the focus of control of the system. In a design diagram for real-time embedded systems, the activation may be bound to a processor and the activation will be given a concrete schedule on this processor (see [KS01]). In such a context, an activation is also bound to a concrete execution sequence, either modeled in a statechart diagram or as program code.

While changes in life state of an object (i.e., creation or destruction) are easily expressible in sequence diagrams, it may sometimes be supportive to depict other changes in object states as well without linking to an explicit state machine. Assigning transitions (or states) to activation symbols may be a means to achieve this. Since the distinction between live activations and periods where an activation is actually computing (see [OMG01], section 3.63.3) cannot always be expressed based on the corresponding messages, e.g. when the scheduling of processes is involved the same concept can be applied to denote execution states of activations.

As another example, in the context of multimedia applications we have identified the necessity not to simplify activations of objects in sequence diagrams to a black-box reaction to some received message (see e.g. [SE99]). The invocation of the presentation of a multimedia object may cause a sequence of internal actions for the receiver or may itself be a complex pattern of interaction (compare Sect. 5). Therefore, we like to have activations as individual modeling elements, in the meta model that allow the modeler to define semantics for activations, and to specialize them for particular application domains by stereotypes.

Although some of these requirements could be fulfilled by adequately combining sequence and statechart diagrams, UML users often tend to minimize the number of different partial models and thus the problems of consistency and view integration.

Hence in general it may be discussed whether sequence diagrams should depict internal, non-observable behavior. Nevertheless, under certain circumstances, activations can have properties of their own that are to be expressed in the model. They may then also occur in constraint expressions. If activations are considered to be meaningful, a consistent representation on all levels of the UML specification is needed. Once activations are semantically present, they may be used to distinguish different kinds of internal activity, possibly leading to a further refinement of this dimension.

9 Defining the Semantic Space of Sequence Diagrams

After the previous discussion and the identification of modeling aspects and their semantics, we will now set out to define the semantic space of sequence diagrams. Figure 2 gives an overview of the dimensions identified in the preceding sections. For rigorous modeling, it has to be stated whether the diagram should be interpreted to

- specify required, forbidden or mandatory behavior (scope),
- contain instances or roles (abstraction),
- display only interactions or internal actions as well (function view),
- include a composition of activations, structure or whole diagrams (composition),
- impose a partial order or a total order on its elements (ordering),
- quantify the temporal order of its elements by an implicit or explicit time metric (time quantification).

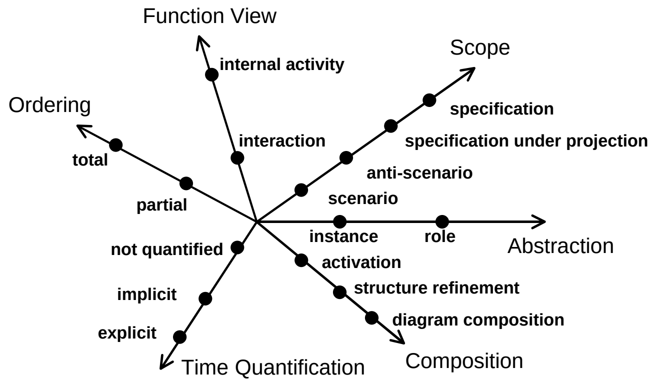

Fig. 2: Dimensions of the semantic space of sequence diagrams and their semantic alternatives

Note that these are just the general decisions to be undertaken. Most of the choices require additional specifications to become really precise (e.g. imposing a total order by the vertical position in the diagram).

As already stated in the previous sections, not all of these semantic concepts are supported by the current UML semantics description and hence the subspace of the supported concepts can be identified (see fig. 3). This reduces some of the dimensions to mere points (see e.g. composition that is not supported in UML). But still two dimensions remain, where a mismatch of UML's syntax and semantics can be detected. These are marked with grey shading.

Based upon this observation two discussions may ensue: On the one hand there seem to be several semantic concepts for sequence diagrams that are currently not supported by the UML. Nonetheless these concepts do not seem to be very exotic or esoteric, as they are frequently used in practice. Thus an inclusion of these concepts in the UML standard may be discussed. On the other hand, there are severe insufficiencies in the relation of syntax and semantics of the UML. For example, if an activation is given a meaning (see [OMG01], section 3.62.1), but there is no element of the abstract syntax to represent an activation, the meaning can never be formalized and fixed. The concept of activations remains vague. Furthermore, there are no syntactic means to distinguish between scenarios and specification diagrams. These missing elements may be supplied by defining stereotypes to indicate the se

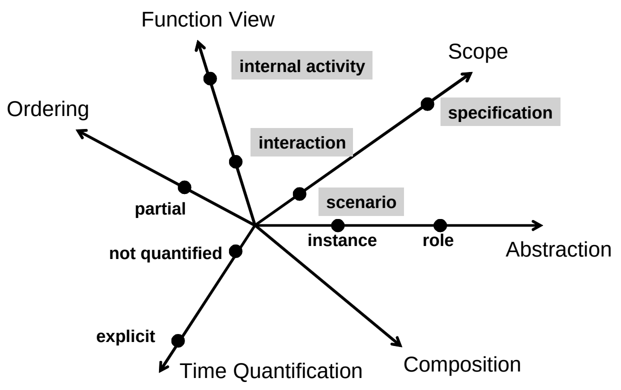

Fig. 3: Semantic space of sequence diagrams as currently supported by the UML

10 Conclusion and Future Work

In this paper, we have developed a semantic space for sequence diagrams. This semantic space consists of dimensions where each dimension can be considered a semantic variation point of sequence diagrams. Semantic variation points have been identified by surveying sequence diagrams in general and UML sequence diagrams in particular. The purpose of the survey was to find semantic modeling concepts existent in sequence diagrams. We have shown that UML sequence diagrams currently have an ambiguous semantics with respect to several of the semantic variations identified. For each semantic variation point, we have identified the choices of semantics and we have justified each choice by supplying examples for its usage in sequence diagram modeling.

Concerning the precision of sequence diagrams, it is necessary that modelers are able to indicate the semantics of sequence diagrams with respect to semantic variations. In order to do this, we propose a method consisting of two steps. First, one has to define the semantic framework by introducing stereotypes for each of these semantic variations. Second, when using a sequence diagram, the modeler has to supply one stereotype for each dimension, thereby fixing the semantics. Each combination of stereotypes defines a new semantics for sequence diagrams. For each combination, specific well-formedness rules may be supplied. As a consequence, the definition of stereotypes for each variation is an important task and should be done with great care. Essentially, this task corresponds to defining a new specialized sublanguage. The use of stereotypes for fixing sequence diagram semantics can easily be integrated in CASE tools such as Rational Rose or TogetherJ. Thus it can be ensured that the semantics of each sequence diagram can be fixed by the modeler.

From a methodological point of view, it needs to be discussed whether all combinations of semantic variations fit together. Using OCL constraints, these invalid combinations can be specified and excluded from the set of all valid sequence diagrams.

Within a rigorous development process, it is of importance that modeling activities can be assigned with sequence diagrams with specific semantics. For example, it may be possible to use scenarios in early stages (e.g. requirements engineering) and then proceed to specifications in later stages (e.g. analysis and design). Using our method to fix the semantics of sequence diagrams, a process model can now precisely define the form of sequence diagram to be used, e.g. by restricting or prescribing stereotypes for a development activity. Due to the fixed semantics, precise consistency relations and checks [EHK01] can be formulated that have to hold between different (partial) models within a development process. Thus it can be ensured that sequence diagrams are used in a way that no contradictions occur.

With respect to further evolving the UML, our discussion can provide input to the definition of semantics and the integration of languages such as MSCs. Using our semantic space, other dialects similar to sequence diagrams can be evaluated and the space spanned in the semantic space can be determined. On the basis of this result, it can then be discussed which semantic elements should be integrated into the UML. We see our results as a basis for both ensuring the general applicability of UML sequence diagrams and simultaneously improving its preciseness by the explicit distinction of syntactic and semantic variations.

Acknowledgement

We want to thank our dear colleague Reiko Heckel for fruitful discussions and valuable suggestions on the topics of this paper.

Bibliography

[BJR99] Booch, G.; Jacobson, I.; Rumbaugh, J.: Unified Software Development Process. Addison-Wesley, Reading, MA, 1999.

[DH98] Damm, W.; Harel, D.: LSCs: Breathing Life into Message Sequence Charts. Technical Report CS98-09, Weizmann Institute of Science, Faculty of Mathematics and Computer Science, January 1998.

[EHK01] Engels, G; Heckel, R.; Küster, J.M.: Rule-Based Specifications of Behavioral Consistency based on the UML Meta-Model. In Proceedings of the Fourth International Conference on the Unified Modeling Language, Toronto, Canada, 2001. To appear.

[FR99] France, R; Rumpe, B. Eds.: Proceedings of UML'99—Beyond the Standard, volume 1723 of Lecture Notes in Computer Science. Springer-Verlag, 1999.

[HHS01] Hausmann, J.H.; Heckel, R.; Sauer, S.: Towards Dynamic Meta Modeling of UML Extensions: An Extensible Semantics for UML Sequence Diagrams. In Proceedings of the IEEE Symposia on Human-Centric Computing Languages and Environments (HCC'01), Stresa, Italy, 2001. To appear.

[ITU96] ITU-TS: ITU-TS Recomendation Z.120: Message Sequence Chart 1996 (MSC96). Technical report, ITU-TS, Geneva, 1996.

[Kn99] Knapp, A.: A Formal Semantics of UML Interactions. In (France and Rumpe Eds.) [FR99]; pp. 116–130.

[KS01] Küster, J.M.; Stroop, J.: Consistent Design of Embedded Real-Time Sytems with UML-RT. In Proceedings of the 4th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'2001), 2001.

[MW01] Mehner, K.; Weymann, B..: Visualization and Debugging of Concurrent Java Programs with UML. In (De Pauw, W; Reiss, S.P.; Stasko, J.T. Eds.): Proceedings of ICSE 2001 Workshop on Software Visualization, Toronto, 2001; pp. 59–64.

[OMG99] Object Management Group: UML 2.0 RFI. Document ad/99-08-08, 1999. http://cgi.omg.org/techprocess/meetings/schedule/UML_2.0_RFI.html

[OMG01] Object Management Group: UML Specification, version 1.4. Document ad/01-02-13, May 2001. http://www.omg.org/

[Oe99] Oevergaard, G.: A Formal Approach to Collaborations in the Unified Modeling Language. In (France and Rumpe Eds.) [FR99]; pp. 99–115.

[SE99] Sauer, S.; Engels, G.: Extending UML for Modeling of Multimedia Applications. In (Hirakawa, M.; Mussio, P. Eds.): Proceedings of the IEEE Symposium on Visual Languages (VL'99), Tokyo, Japan, 1999; pp. 80–87.

[WD99] Wills, A.C.; D'Souza, D.F.: Objects, Components, and Frameworks with UML: The Catalysis Approach. Addison-Wesley, Reading, MA, 1999.