The many meanings of UML 2 Sequence Diagrams: a survey[*†]

Zoltán Micskei[1] and Hélène Waeselynck[2,3]

[1]Budapest University of Technology and Economics, Muegyetem rkp. 3, Budapest 1111, Hungary
[2]CNRS; LAAS, 7 avenue du Colonel Roche, 31077 Toulouse, France
[3]Université de Toulouse; UPS, INSA, INP, ISAE; LAAS, 31077 Toulouse, France

2011

Abstract

Scenario languages are widely used in software development. Typical usage scenarios, forbidden behaviors, test cases, and many more aspects can be depicted with graphical scenarios. Scenario languages were introduced into the Unified Modeling Language (UML) under the name of Sequence Diagrams. The 2.0 version of UML changed Sequence Diagrams significantly and the expressiveness of the language was highly increased. However, the complexity of the language (and the diversity of the goals Sequence Diagrams are used for) yields several possible choices in its semantics. This paper collects and categorizes the semantic choices in the language, surveys the formal semantics proposed for Sequence Diagrams, and presents how these approaches handle the various semantic choices.

1 Introduction

Scenario languages are widely used in software development. Typical usage scenarios, forbidden behaviors, test cases, and many more aspects can be depicted with graphical scenarios. Several language variants were proposed over the years. The International Telecommunication Union's (ITU) Message Sequence Chart (MSC) [ITU11] was one of the first of such languages. It is widely used, since its first introduction in 1993 it was updated several times, and the specification

[*]This work was partially supported by the ReSIST Network of Excellence (IST 026764) funded by the European Union under the Information Society Sixth Framework Programme.

[†]The final publication is available at Springer via `http://dx.doi.org/10.1007/s10270-010-0157-9`

defines also a formal semantics for the basic elements of the language based on process theory. Triggered message sequence charts (TMSC) [SC06] proposed extensions to MSC to express conditions and refinement in a precise way. Live Sequence Charts (LSCs) [DH01] concentrated on distinguishing possible and necessary behaviors. A special technique and a tool, the Play-Engine, were also developed for LSC to specify reactive systems [HM03].

Scenario languages were introduced into the Object Management Group's (OMG) Unified Modeling Language (UML) under the name of Sequence Diagrams. The 2.0 version of UML changed Sequence Diagrams significantly. Several elements were borrowed from MSC, many new complex elements were added to the language, and the semantics and the underlying metamodel were rewritten. Due to the increased expressiveness of the language, interpreting a complex diagram that uses the new constructs is a difficult task; thus, having a precise formal semantics becomes even more critical. But the many different purposes Sequence Diagrams are used for, e.g., showing the flows of method calls inside a program, or giving a partial specification of interactions in a distributed system, require quite different interpretations of the language. Indeed, many different semantics have been proposed for Sequence Diagrams. For a practitioner wanting to use Sequence Diagrams for a given purpose, it is not easy to select a suitable semantics.

We faced exactly this problem when we were working on the definition of test languages for mobile computing systems. When we tried to define the semantics of the new language, we encountered the problem that the various formal semantics for Sequence Diagrams handle even the most basic diagrams quite differently. It turned out that there are several subtle choices in the interpretation of language constructs. Moreover, these choices and all their consequences are often not obvious. A structured representation of all these choices was needed.

Based on our experience, our aim was to (i) give an overview about the proposed *formal semantics*, (ii) *collect and categorize* the semantic choices faced by them, and (iii) present the *different options* for the collected choices and the relations between these options in a structured format.

The paper is divided into the following parts. Section 2 presents the syntax and semantics as defined in the OMG specification. We tried to highlight those parts, which are usually missing from published overviews about Sequence Diagrams. Section 3 presents a survey of 13 proposed for- mal semantics for Sequence Diagrams. Section 4 collects and categorizes the semantic choices in the existing formal semantics and describes how the different approaches handle them. Section 5 illustrates the insights gained from the survey, by taking TERMOS as a case study. We show how the identification of where the choices are, and which options can be taken, proved helpful to devise a semantics well-suited for the intended usage of this language.

2 UML Sequence Diagrams in the OMG specification

Sequence Diagrams are defined in the UML Superstructure specification [OMG11]. More precisely, scenarios in UML are modeled with *Interactions*[1]. Interactions can be illustrated on several diagram types: Sequence Diagrams, Interaction Overview Diagrams, Communication Diagrams, Timing Diagrams, and Interaction Tables. Thus, the syntax and semantics are defined for Interactions; Sequence Diagrams are just a concrete notation to depict them.

2.1 Syntax of Sequence Diagrams

The syntax defined in the specification consists of (i) a concrete syntax defining the graphical notation, and (ii) an abstract syntax given with a metamodel defining the relationships between the elements.

2.1.1 Concrete syntax

This section summarizes the elements of Interactions and their notations on Sequence Diagrams. Figure 1 illustrates a basic *Interaction*. *Lifeline*s represent the individual participants in the Interaction, which communicate via *Message*s.

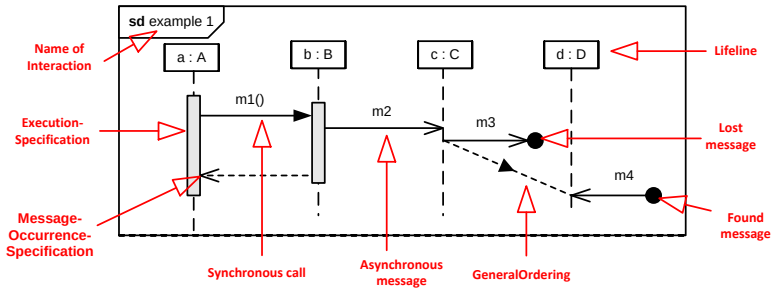


Figure 1: Example Sequence Diagram

Message is a general term: it can be a synchronous or an asynchronous communication; it can mean calling an *Operation* or sending a *Signal* (specified by its *MessageSort* attribute). *MessageKind* defines whether the sender or receiver of the message is known (complete, lost or found messages). Messages have two *MessageEnd*s. *GeneralOrdering* can constrain the ordering of otherwise unrelated occurrences. *ExecutionSpecification* is a specification of the execution of a unit of behavior or action within a Lifeline. *OccurrenceSpecification* (and its descendants) is the basic unit of semantics. Sending and receiving messages are marked with *MessageOccurrenceSpecification*; starting and ending of ExecutionSpecifications are represented with *ExecutionOccurrenceSpecification*.

More complex Interactions can be created with *CombinedFragment*. A CombinedFragment consists of one or more *InteractionOperand*s. An *InteractionOper-*

[1]Throughout the text, elements of the UML metamodel are written with CamelCase.

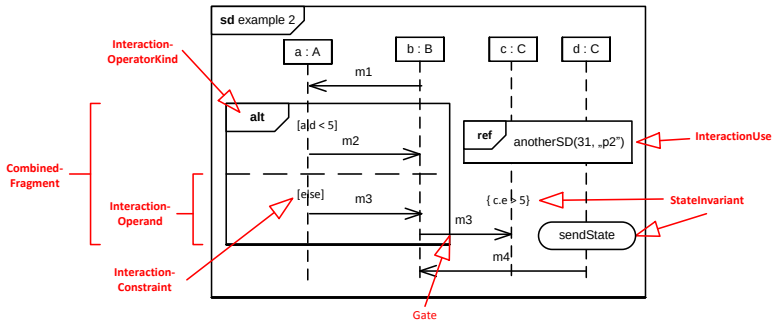


Figure 2: Example for CombinedFragment

atorKind specifies the purpose of the fragment. *InteractionConstraint*s can guard each InteractionOperand. Messages on their own cannot cross the boundaries of CombinedFragments: they need a *Gate* which links the two parts of the message. An *InteractionUse* refers to another Interaction. It can be passed parameters and can have a return value.

StateInvariant is a run-time constraint on one of the participants of the Interaction. StateInvariants have two kinds of notation: it can be an expression of attributes and variables, or it can refer to a state of the Lifeline's instance (both notations are used on Figure 2). Further constructs exist, e.g., for specifying time and duration constraints, for a complete list see the specification [OMG11].

2.1.2 Abstract syntax

The abstract syntax of Interactions is defined with metamodeling; the model is presented in *Section 14.2 Abstract Syntax* of [OMG11]. The abstract syntax is depicted in several separate diagrams, which makes it sometimes hard to see all the connections between the important elements. Thus, we illustrate the abstract syntax of the most important elements of the BasicInteractions package on one diagram in Figure 3. (Note that the various Events classes, the MessageSort and MessageKind classes, attributes and some of the association names are not depicted on the picture for readability.)

Figure 4 illustrates the abstract syntax of the Fragments package. (Again, attributes and some of the association names are not depicted on the picture for readability.) *InteractionFragment* is an abstract class for Interaction, CombinedFragment, InteractionOperand, InteractionUse and Continuation, and also for OccurrenceSpecification, ExecutionSpecification and StateInvariant.

From the abstract syntax we can see for example, that a StateInvariant belongs to one Lifeline; thus it is a local constraint, or that there are three kinds of Gates, each for different purposes.

The specification contains a simple example illustrating an Interaction's concrete and abstract syntax; however that diagram does not contain CombinedFragments. It is really helpful to see how the different elements relate to each other;

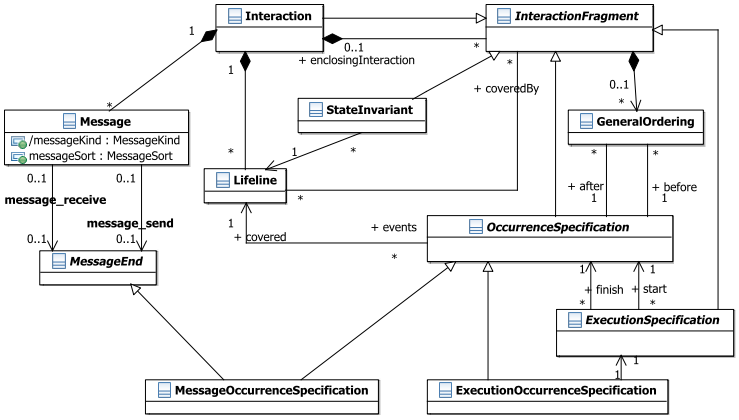


Figure 3: The abstract syntax of the BasicInteractions package (fragment)

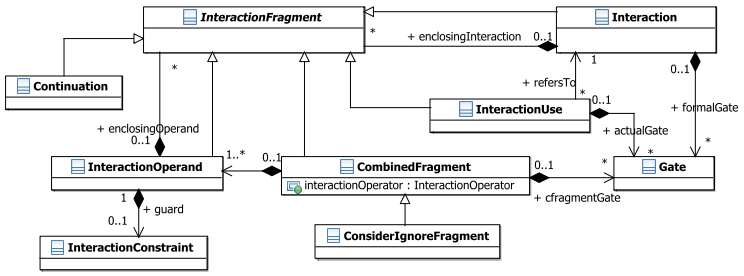


Figure 4: The abstract syntax of the Fragments package (fragment)

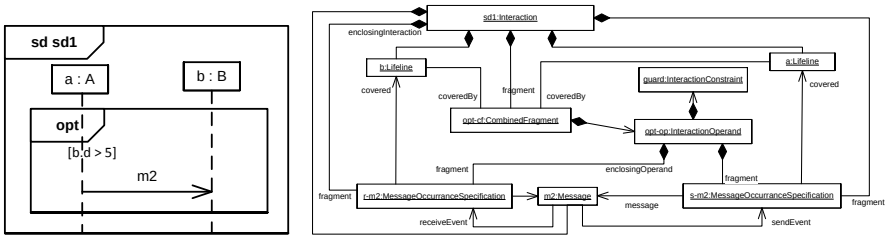
thus a more complex example is included here.

The right side of Figure 5 contains the metamodel elements of *sd1*. The Interaction is a container for all other elements. The OccurrenceSpecifications are linked to the appropriate Lifelines and Messages. The Lifelines are connected to the CombinedFragments that cover them. The InteractionOperand contains the InteractionFragments (OccurrenceSpecifications, StateInvariants, other CombinedFragments, etc.) which are enclosed by this operand. An InteractionFragment can be enclosed only by one operand; thus when an InteractionFragment is nested in several operands, only the bottom-most containment is illustrated in the model explicitly.

2.2 Semantics of Sequence Diagrams

There are two major challenges when dealing with the semantics given in the OMG specification.

- The description of the semantics is *scattered* throughout the text. Some parts are in the introduction of the chapters, while some information is



(a) Concrete syntax (b) Abstract syntax

Figure 5: A complex Interaction's concrete and abstract syntax (fragment)

only in the constraints defined in the detailed description of a class.

- The specification uses so-called *semantic variation points* [Sel04], i.e., part of the semantics is not specified in detail to allow using UML in many domains. When UML is used in a concrete domain, the modeler has to choose from the different possible variations. However, sometimes these variation points are not marked explicitly.

UML introduced a common run-time semantics for its different notations, which defines basic elements, e.g., Behavior, Action, and Event. Knowledge of this common run-time semantics is helpful, but not necessary to understand the semantics of Interactions. Thus, we will skip it. Interested readers could find it in the CommonBehaviors package of the specification or in [Sel04], where not only the concepts, but also the design decisions behind them are explained.

This section summarizes the parts of [OMG11] that deal with the semantics of Interactions. (Note, in the remaining part of this section page numbers refer to [OMG11].)

2.2.1 Semantics of basic Interactions

Interactions describe behavior with messages between participants. The focus is on the order and the types of the messages, although Interactions can contain reference to data in message parameters and constraints. The central concept of the semantics is a trace.

> "A trace is a sequence of event occurrences, each of which is described by an OccurrenceSpecification in a model" (page 495).

A central question is what part of the behavior is modeled by the Interactions.

> "There are normally other legal and possible traces that are not contained within the described interactions" (page 473).

Interactions can model also invalid traces, and there could be traces that are not described by the Interaction:

> "The semantics of an Interaction is given as a pair of sets of traces. The two trace sets represent valid traces and invalid traces. The union of these two sets need not necessarily cover the whole universe of traces. The traces that are not included are not described by this Interaction at all, and we cannot know whether they are valid or invalid" (page 495).

Collecting all the references yields that invalid traces are defined by assert and negative fragments, and constraints such as StateInvariant, DurationConstraint and TimeConstraint. The semantics of Interactions is explained with an interleaving semantics, i.e., two events may not occur at exactly the same time.

Producing the traces of a diagram is constrained by the following rules:

- Occurrences on the same Lifeline must occur in the same order as they are specified, even for the receiving of messages sent by different objects (page 483).
- Receiving a message should occur after the sending of the message (page 507).
- GeneralOrdering can add further constraints to OccurrenceSpecifications, which are not related.

Thus the semantics defines partial orders on OccurrenceSpecifications, and valid traces are those, which can be generated satisfying these orders.

2.2.2 Semantics of fragments

If no operator is explicitly given, then the InteractionFragments of a diagram should be combined using a form of sequential composition, weak sequencing (page 500). As Figure 4 shows, OccurrenceSpecifications are also InteractionFragments; thus this default composition applies also to basic Interactions. The rules for weak sequencing are the following (page 483):

1. "The ordering of OccurrenceSpecifications within each of the operands is maintained."
2. "OccurrenceSpecifications on different lifelines from different operands may come in any order."
3. "OccurrenceSpecifications on the same lifeline from different operands are ordered such that an OccurrenceSpecification of the first operand comes before that of the second operand."

The other sequencing construct, strict sequencing, has a stronger version of the second rule: OccurrenceSpecifications on different Lifelines from different

operands become ordered as in the third rule (that is, the content of the first operand comes before that of the second operand).

For the Interactions that use elements from the Fragments package, the semantics is mainly defined in the description of the CombinedFragment element when detailing the various operators (pp. 482–485). We grouped the operators into the categories of Table 1.

The first category contains operators that introduce choice and iteration. The operators in the second category are for parallelization and sequencing. Operators in the last category are related to the conformance relation, i.e., the way a trace is categorized as valid, invalid or inconclusive according to a diagram. For example, an *assert* describes a mandatory behavior, while a *neg* one that should not happen. The operators *consider* and *ignore* change the set of message names from which valid and invalid traces can be built (see later in Section 4.5.2).

Other important classes defined in the Fragments package are *InteractionConstraint* (guards on CombinedFragment) and *variables* (local attributes and parameters of Interactions, arguments of Messages).

The semantics presented in the OMG specification gives a basic idea how Sequence Diagrams should work. However, this natural language semantics is incomplete and ambiguous; thus we need to look into existing formal semantics to understand how Sequence Diagrams are interpreted in practice.

3 Overview of proposed semantics

Many formal semantics were proposed for UML 2 Sequence Diagrams over the years. We selected thirteen approaches, listed in Table 2. As the UML 2.0 specification completely changed how Interactions are defined (different semantics, introduction of invalid traces and CombinedFragments, etc.), the table does not contain approaches for UML 1.x Sequence Diagrams.

There are many other papers proposing a semantics for UML 2 Sequence Diagrams (e.g., [Bro+08; Cen07]), and it is impossible to include all of them. The selected 13 approaches contain both pioneering works which influenced most of the others, and less referenced ones which concentrated on specific usages of Sequence Diagrams. It is thus hoped that they are representative for the different possible choices and options, at least to some extent.

Interested readers can find detailed examples for using some of the semantics on an example diagram in our technical report [MW08].

Table 3 collects which constructs are mentioned in the different approaches. Note that the different approaches sometimes redefine the meaning of the original constructs, and handle the given elements at very different levels of detail. Thus, the goal of this table is not to calculate a percentage of how much of the specification is covered by each work; instead, it may serve as a reference to search which publication mentions a given element.

Table 1: Operators in CombinedFragment

Operators that introduce choice and iteration	
alt	"alt designates that the CombinedFragment represents a choice of behavior."
opt	"opt designates that the CombinedFragment represents a choice of behavior where either the (sole) operand happens or nothing happens."
break	"break designates that the CombinedFragment represents a breaking scenario in the sense that the operand is a scenario that is performed instead of the remainder of the enclosing InteractionFragment."
loop	"loop designates that the CombinedFragment represents a loop. The loop operand will be repeated a number of times."
Operators for parallelization and sequencing	
par	"par designates that the CombinedFragment represents a parallel merge between the behaviors of the operands."
seq	"seq designates that the CombinedFragment represents a weak sequencing between the behaviors of the operands."
strict	"The semantics of strict sequencing defines a strict ordering of the operands on the first level within the CombinedFragment."
critical	"critical designates that the CombinedFragment represents a critical region. A critical region means that the traces of the region cannot be interleaved by other OccurrenceSpecifications."
Operators that are related to the conformance relation	
neg	"neg designates that the CombinedFragment represents traces that are defined to be invalid."
assert	"assert designates that the CombinedFragment represents an assertion. The sequences of the operand of the assertion are the only valid continuations. All other continuations result in an invalid trace."
ignore	"ignore designates that there are some message types that are not shown within this combined fragment."
consider	"consider designates which messages should be considered within this combined fragment."

Table 2: Summary of proposed semantics

Name	References	Years	Comments/Tools
Störrle	[Stö03a; Stö03b; Stö04]	2003–2004	
STAIRS	[HS03; Hau+05; RHS05a; RHS05b; LS06; Run07; Lun08]	2003–2008	Implemented in Maude
Cavarra & Filipe	[CK04a; CK05a]	2004	
Cengarle & Knapp	[CK04b; CK05b; CGW06; CK08]	2004–2008	
Küster-Filipe	[Küs06; Bow06]	2005–2006	
P-UMLaut	[Eic+05]	2005	P-UMLaut tool
Grosu & Smolka	[GS05]	2005	
Hammal	[Ham06]	2006	
MSD	[HKM07; HM08]	2006–2008	Synchronous, S2A tool
Knapp & Wuttke	[KW07]	2006–2007	HUGO/RT tool
Thread-tag based	[DHC07]	2007	
CPN	[Fer+07]	2007	Synchronous
Template semantics	[SVN08a; SVN08b]	2008	

Table 3: Overview of the mentioned elements in each approach

	Störrle	STAIRS	Cavarra & Filipe	Cengarle & Knapp	Küster-Filipe	P-UMLaut	Grosu & Smolka	Hammal	MSD	Knapp & Wuttke	Thread-tag	CPN	Template semantics
Interaction													
Local attributes		•				•							
GeneralOrdering	•			•		•				•	•		
Message													
argument		•				•				•			
asynchronous	•	•	•	•	•	•	•	•		•	•	•	•
lost, found					•	•							
creation, destruction						•							
CombinedFragment													
guard		•	•		•	•		•	•	•			•
alt	•	•	•	•	•	•	•	•	•	•	•	•	•
opt	•	•		•		•		•	•	•	•	•	•
loop	•	•		•		•	•	•	•	•	•	•	•
break	•					•		•	•				•
par	•	•	•	•	•	•		•		•	•	•	•
seq	•	•	•	•	•	•	•	•	•	•	•	•	•
strict	•	•	•	•		•	•	•	•	•	•		
critical	•					•							•
neg	•	•	•	•	•		•		•	•			
assert	•	•	•	•	•				•				
ignore	•			•					•	•			
consider	•			•					•				
Other elements													
Gate		•			•	•							
StateInvariant		•	•		•	•			•	•			
DurationConstraint	•	•						•					
TimeConstraint	•	•						•					
InteractionUse (ref)	•	•			•	•	•						
argument						•							

If we look through the table, the following observations can be made:

- Conformance-related operators were not considered in one third of the approaches. Even if it is one of the most important aspects of the language, it is hard to formalize it and solve its issues. Moreover, *consider* and *ignore* were not mentioned in four of the eight that dealt with conformance.
- Gates were handled explicitly in only a small number of papers.
- Variables and arguments were also not mentioned in several approaches. It is understandable, because they are not in the focus of Sequence Diagrams, and not easy to express in some of the formalisms.
- Handling time and time constraints was also not common.
- Some of the elements (ExecutionSpecification, Continuation, PartDecomposition) were not explicitly handled in any of the approaches; thus we left them out from the table.

The rest of the section gives a brief description of each of the approaches. As there are 13 approaches, the overall content is quite long. Readers more interested in the different semantics choices can jump directly to the discussion in Section 4, and return to some of the approaches later.

3.1 Trace semantics from Störrle

Störrle was one of the firsts to propose a semantics for UML 2 Sequence Diagrams in [Stö04] (previously published in [Stö03a; Stö03b]). It is a trace-based semantics, which contains much of the elements of the OMG specification. The semantics defined the set of valid and invalid traces for "plain InteractionFragments", i.e., ones without CombinedFragment. Later, for CombinedFragment the semantics of each operator is presented. At that time, the OMG specification was still in a draft version; since then a few element names have changed. Section 3.1 in [Stö04] analyzes the semantic approach used in the OMG specification and finally categorizes it as an interleaving, linear-time semantics of complete traces with abstract real time. Section 5 in [Stö04] deals with *assert* and *neg* in detail, and gives several potential meanings for the *neg* operator. It also points out many issues with the OMG specification.

3.2 STAIRS approach

In [HS03] the authors introduce STAIRS (Steps To Analyze Interactions with Refinement Semantics). They define a denotational, trace-based semantics for Sequence Diagrams, where the focus is on the precise definition of refinement for Interactions. Three types of refinement are defined:

- *Supplementing*: inconclusive traces are categorized as either positive or negative;
- *Narrowing*: some of the positive traces are categorized now as negatives;
- *Detailing*: introducing a more detailed description without significantly altering the externally observable behavior.

In [Hau+05] the approach is extended to Timed STAIRS; the semantics is modified in a way that the reception and consumption of messages are differentiated (this leading to three event types: transmission, reception, consumption). In [RHS05a] the interpretation of the neg operator is analyzed, and new operators ($refuse$, $veto$) are proposed instead of it. The dissertation [Run07] summarizes the denotational STAIRS and its extensions.

In [LS06] (and later greatly extended in [Lun08]) an operational semantics is given complying with the above denotational semantics. In Section 7.3 of [Lun08] a good overview is given of the challenges when defining semantics for UML Sequence Diagrams. The operational semantics uses a reduced abstract syntax given by a grammar to represent Sequence Diagrams. The model of the operational semantics consists of an execution system, which stores the state of the communication channels and the sequence diagram, and a projection system, which finds the enabled events. The operational semantics is also implemented in the Maude language.

3.3 ASM-based semantics of Cavarra & Filipe

In [CK05a] the authors proposed a technique using Object Constraint Language (OCL) templates to express liveness properties in UML Sequence Diagrams, based on results of LSC [DH01]. Using concepts from LSC, several problematic parts of the OMG specification were addressed. May and must behavior, universal, and existential diagrams can be differentiated. In Fig. 2 in [CK05a] the authors give a nice example that certain liveness properties cannot be expressed with $assert$ or neg. Therefore, they propose an after/eventually OCL template, which says that after a condition becomes true there is a guarantee that eventually another condition will become true. Moreover, they introduce global constraints and methods for synchronization at the beginning or end of CombinedFragments.

In [CK04a], the authors defined a semantics to this liveness-enriched Sequence Diagrams using abstract state machines (ASM). Locations are associated with each important point on the Lifelines. For each instance, a separate process is assigned. ASM rules are defined to specify the progress of one instance depending on what kind of fragment the instance currently is in. In the conclusion, several good observations are made on the challenges of UML Sequence Diagrams.

3.4 Trace-based semantics of Cengarle & Knapp

In [CK04b] the authors define a denotational semantics for the traces of Interactions using pomsets (partially ordered multisets). Later, in [CK05b] an operational semantics is given for Sequence Diagrams. The semantics of the positive fragments is similar to the one defined by Störrle. The authors concentrate on the interpretation and definition of negative fragments. Rules are given for each of the operators specifying whether a trace positively or negatively satisfies a fragment with that operator. The authors point out that with the basic interpretation of negative fragments it is easy to construct *overspecified* Interactions, i.e., an Interaction that can be positively and negatively satisfied from the same trace. In the paper [CK04b] the operator not is introduced instead of neg and $assert$ to overcome some of the problems with negative satisfaction. Later, the work is extended in [CK08] to define the semantics using a different formalism (namely institutions), and in [CGW06] to handle variability expressed on a diagram.

3.5 True-concurrency semantics from Küster-Filipe

Küster-Filipe defined a true-concurrent semantics based on event structures in [Küs06]. In [Bow06] the semantics is extended to handle the InteractionUse construct. It considers only a smaller number of operators and constructs (alt, par, seq, and StateInvariant), but gives them a well-defined semantics.

The semantics uses the temperature (hot and cold messages) concept from LSC to express mandatory or possible behavior. For example, hot messages *must* be received, while cold messages *may* be received after sending. Furthermore, it uses LSC's location concept to mark occurrences on a Lifeline.

The approach constructs for every Lifeline a labeled prime event structure. The model takes into account the possible nesting of CombinedFragments and gives a very clear definition for the predecessors of every event. Finally, the event structures for the different Lifelines are combined according to the Messages sent between them. In the end of [Küs06] a two-level temporal logic is presented, which can be used to specify Interactions.

3.6 M-net based semantics of the P-UMLaut project

In [Eic+05] a semantics is given for Sequence Diagrams based on M-nets (multi-valued nets), which is an algebra based on high-level Petri nets. The method handles basic data types (Boolean and integers); thus, it can include the local attributes of Interactions, the arguments of Messages, and the evaluation of conditions in the semantics. M-net fragments are given for basic constructs, like starting of a Lifeline or sending and receiving of a message. These are then connected by composition operators according to the enclosing CombinedFragment's operator. The semantics defined in the paper assumes that all behavior is explicitly specified in the diagrams and no conformance-related operator is used.

3.7 Safety-liveness semantics from Grosu & Smolka

In [GS05] the authors propose to interpret valid and invalid parts of an Interaction as liveness and safety properties, respectively. The Sequence Diagrams are first transformed to hierarchic, non-deterministic automata, then the high-level automata are flattened, and finally liveness Büchi automata are constructed from the positive automata, and safety Büchi automata from the negative ones. Based on the languages these automata accept, refinement of Sequence Diagrams is defined.

The paper only treats the combination of basic diagrams with no CombinedFragment and bounded high-level Interaction Overview Diagrams. In this way, their trace language is regular, but it is a restriction of the OMG specification.

3.8 Branching time semantics from Hammal

The author of [Ham06] presents a denotational semantics based on partial orders. It assigns to each fragment a graph containing the OccurrenceSpecifications and their relations. The structures are later enriched with timing information using the timing constraints on the diagram.

3.9 Modal Sequence Diagrams

Modal Sequence Diagrams (MSD) [HKM07; HM08] are an extension to UML Sequence Diagrams by Harel and Maoz, which adapts LSCs to the notation of UML. LSC is a language inspired from MSC that allows the specification of possible and mandatory scenarios.

The authors point out that the root of all the challenges regarding $assert$ and neg are that these were introduced as simple operators, while they are rather modalities. UML Sequence Diagrams do not have a clear definition of the modalities of the diagrams and thus the authors apply the model of LSC to UML. The *modal* stereotype is attached to InteractionFragments to specify whether it describes a hot (universal) or cold (existential) behavior. A hot fragment represents a behavior that is mandatory, while the cold represents only a possible behavior. The operators $assert$ and neg are used then just as syntactic notation to show whether the constructs inside them have hot or cold modality. The authors also treat the question how multiple diagrams should be handled, one point that is often missing from others.

In the Appendix, a formal semantics based on weak alternating automata is sketched. First, the diagram is transformed into an intermediate format, an unwinding structure, from which the states (the cuts of the diagrams) and the transitions (message sending) of the automaton are derived. The current semantics considers only synchronous messages; the sending and receiving is treated as one event.

3.10 Operational semantics from Knapp & Wuttke

The paper [KW07] proposes an operational semantics, where an interaction automaton is produced by unwinding the Interaction. One single interaction automaton is created for the entire Interaction. The authors apply some restrictions to ease the processing of Sequence Diagrams (e.g., replace neg with the binary logic variant not introduced in [CK04b], restrict the use of not only to basic interactions, restrict loops to only allow basic interactions, etc.). Later, this interaction automaton is used as an observer process in the SPIN model checker to check the communication produced by UML State Machines.

3.11 Thread-tag based semantics

In [DHC07] a trace semantics was proposed for specifying object-oriented programs with multiple threads on the same Lifelines. The authors claim that if the instances of the Interaction are multi-threaded objects, then the ordering should not be specified for messages originating from the same Lifeline; instead, only for those messages which are from the same Lifeline and from the same thread of the Lifeline. For this reason, they extend messages with "thread tags", i.e., identifiers specifying which the sender and receiver threads for that message are. Later, a *trace-based semantics* is given for the operators, where the ordering rules are defined with respect to thread tags. Conformance-related operators are not considered in the paper.

In our opinion, some of the problems presented in the paper can be solved without modifying the original semantics with the help of inline PartDecompositions, i.e., when an instance is decomposed to multiple Lifelines representing its inner connectable elements, like the threads of an object.

3.12 Semantics based on CPN

In [Fer+07] the authors propose a translation that produces a Colored Petri Net from UML use cases and Sequence Diagrams. For the basic operators (opt, alt, par, $loop$, and ref), templates are assigned to show what kind of CPN fragment should be created. The translation does not consider conformance-related operators. It seems, although it is not stated explicitly in the paper, that each diagram contains initially only one active instance (it can later fork into several executions with a par). Only synchronous messages are handled, because the sending and receiving are represented by the same transition.

3.13 Template semantics

In [SVN08a] a formalization using template semantics is proposed for UML 2 Sequence Diagrams. The formalization is described in more detail in the technical report [SVN08b]. The approach gives an operational semantics for which the basic computation model is hierarchical transition systems (HTS). First, the maxi-

mal sequence fragments of the diagram are computed, i.e., the maximal sequences of consecutive Messages that do not contain CombinedFragments. Then, for each Lifeline a complex HTS is formed by composing the maximal blocks of the Lifeline using the InteractionOperators. Finally, the HTSs for the Lifelines are composed using interleaving operators.

4 Semantic choices in Sequence Diagrams

Section 2.2 presented the informal semantics defined in the OMG specification. As it could be seen from the overviews in Section 3, several approaches were proposed to formalize the semantics of UML Sequence Diagrams. This section *collects* and *categorizes* the different choices taken by these approaches. Table 4 presents our categorization of the semantic choices collected.

Table 4: Categorizing semantic choices in UML 2 Sequence Diagrams

Interpretation of a basic Interaction	What is a trace? Categorizing traces Complete or partial traces
Introducing CombinedFragments	Combining fragments
Computing partial orders	Processing the diagram Underlying formalisms Choices and predicates
Introducing Gates	Gates on CombinedFragments Formal and actual Gates
Interpretation of conformance-related operators	Assert and negate Ignore and consider Conformance-related operators in complex diagrams Traces being both valid and invalid

First, the various interpretations of the basic concepts are listed (Section 4.1). Next, the methods for handling the concept of CombinedFragment are analyzed (Section 4.2). Section 4.3 collects how the partial orders of a diagram are computed, and how the related operators and elements (alternatives, guards, etc.) are handled. Section 4.4 is about the different types of Gates. Finally, Section 4.5 details the handling of conformance-related operators.

When necessary, the discussion is illustrated by example diagrams containing traces that show the difference between the options listed in the given section. Since the approaches differ quite heavily in their formalization (basic definitions, symbols to use, etc.), we present each option without its respective formal definition. Some of the subsections do not list every approach, as some UML elements

are not considered by all the approaches.

Furthermore, each subsection ends with a diagram summarizing the different choices and options. Figure 6 illustrates the notation used in these diagrams, which was inspired by feature models [Kan+90]. For example, for A both B and C has to be selected, for D only one of E or F can be chosen, while H is an optional choice, which may or may not be selected. The † symbol marks an option, which departs from the OMG specification. Note, however, that there is no negative connotation associated with this symbol, as several "non-standard" options proved to be really useful for specific applications.

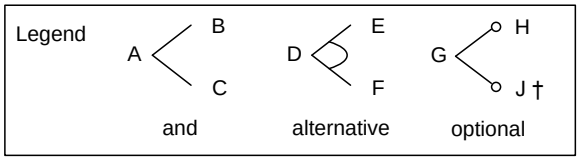


Figure 6: Notation used in the summary diagrams

4.1 Interpretation of a basic Interaction

Let us start the discussion with a simple diagram without any explicit operator. Because the semantics of an Interaction is defined as the valid and invalid traces produced by the diagram, first the content of a trace has to be specified.

4.1.1 What is a trace?

Since the purpose of the semantics is to categorize traces, a definition of traces is needed. Usually to simplify the representation, the notation of $!m1.!m2.?m1.?m2$ is used for traces, where $!m$ denotes sending and $?m$ denotes receiving the message m. However, in a formal semantics, one has to be more explicit, e.g., because there can be several Lifelines in the Interaction sending messages with the same name, it should be specified who sent or received the message.

Thus some of the semantics (STAIRS, Cengarle & Knapp, Grosu & Smolka, Template semantics) represents elements of the trace with tuples, e.g., (action, sender, receiver, message name).

However, on a diagram, where the same message name appears twice between the same Lifelines, the above notation cannot describe the ordering that the receiving of the first m message should come before the receiving of the second one (Figure 7).

Using explicit locations can help this: each OccurrenceSpecification is assigned a unique location name; thus the two receptions of Signal m can be differentiated. The location names are symbolic labels that usually conform to the visual position of the location. Approaches using locations are Störrle, Cavarra & Filipe, Küster-Filipe, P-UMLaut, Hammal, and MSD.

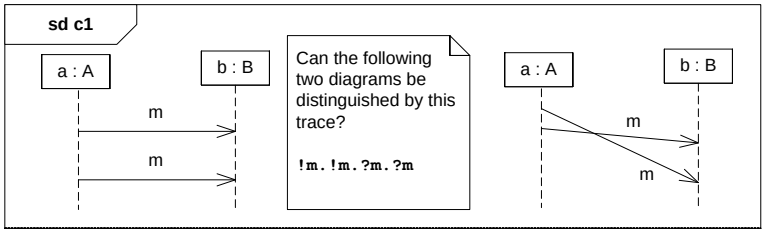
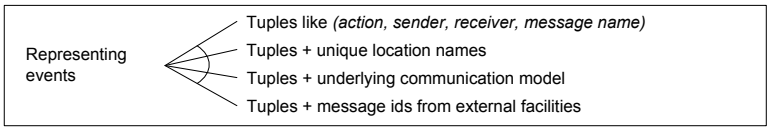


Figure 7: Handling ordering constraints from duplicate messages

Another option can be to specify the underlying communication model. For example, in the operational version of STAIRS, it can be specified whether the execution model should use a global FIFO or one FIFO for each Lifeline, etc.

The above solutions are defined for symbolic traces. In order to analyze concrete system traces, the receiving events should be matched with the sending event that caused it, which is only possible if each message in the trace can be uniquely identified. Thus, each message should have a unique identifier obtained from some external monitoring facility. See, e.g., [Hal+06] for such a definition of a trace, and for applications to monitoring distributed systems.

Note that in our example diagrams, for the sake of simplicity, we will use the shorthand $!m$ instead of $(send, lifeline, m, id)$.



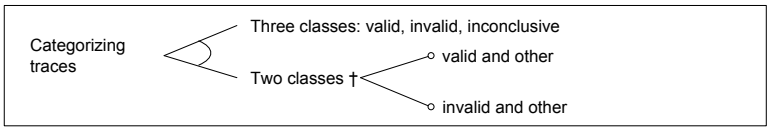
4.1.2 Categorizing traces

Once it is defined how to represent a trace, it should be decided how to categorize the traces. The UML specification gives the semantics of an Interaction as a set of valid and a set of invalid traces. However, it states that there can be other traces, for which we cannot know whether they are valid or invalid.

For the approaches that use Sequence Diagrams for specification and refinement (Störrle, STAIRS, Cengarle & Knapp) using all the three classes is convenient. Usually, they define the valid and invalid traces explicitly, and all other traces are considered inconclusive. Cavarra & Filipe and Küster-Filipe do not explicitly mention inconclusive traces, but they have a separate "aborted" mode for invalid traces; thus they are able to differentiate invalid and inconclusive traces. In MSD, locations and messages have cold or hot temperature assigned. A cold message depicts a potential behavior; thus, a trace violating it is considered as an inconclusive one. A hot message represents a mandatory behavior; its violation results in an invalid trace.

Some approaches differentiate only two classes of traces. The ones using Sequence Diagrams for verification purposes (Grosu & Smolka, Knapp & Wuttke)

separate the traces into either valid/other or invalid/other classes. The focus on validity or invalidity depends on whether the property to be checked is a liveness property (a valid trace is exhibited) or a safety one (no invalid trace is exhibited). The other approaches (P-UMLaut, Hammal, Thread-tag, CPN, Template semantics) are not dealing with conformance-related operators; hence, they do not have invalid traces and may be classified into the valid/other category.



4.1.3 Complete or partial traces

According to the OMG specification, basic Sequence Diagrams specify complete, potential behaviors, meaning that the traces represented by the Interaction are examples for valid traces, and all the other traces are inconclusive with respect to the given diagram. Thus, the standard interpretation of the diagram in Figure 8 is that $!m1.?m1.!m2.?m2.!m3.?m3$ is valid and all other traces are inconclusive. Most of the approaches use this interpretation.

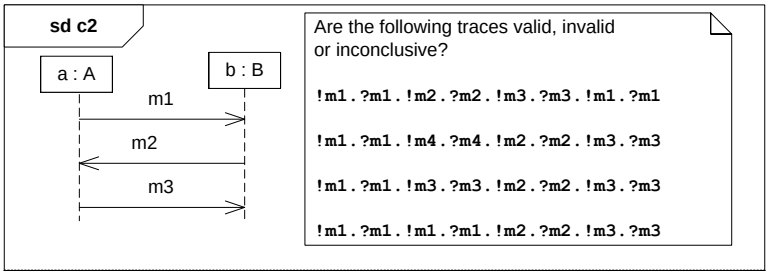


Figure 8: Interpretation of a basic Interaction

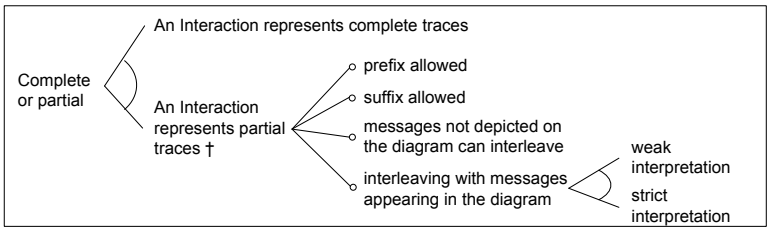
Sometimes this interpretation is not convenient, e.g., when one would like to specify requirements [HM08], safety properties [GS05] or test purposes [Pic03]. For this reason, two of the semantics use an interpretation with partial traces, i.e., the diagram depicts only parts of the valid traces; other messages can interleave with them to form the complete, valid traces. These extra messages usually come from two sources:

- There can be a prefix or suffix for the diagram.
- During the processing of the diagram, messages can interleave with the ones depicted explicitly on it (e.g., messages coming from other Lifelines, other message types not used on the diagram, or duplicate messages).

Grosu & Smolka use an interpretation where a valid trace can have any suffix, but no prefix. In MSD there may be any prefix or any suffix in the system trace before the shown behavior occurs (e.g., in Figure 8 $!m1.?m1.!m2.?m2.!m3.?m3.!m1.?m1$ is a valid trace for both approaches).

For handling interleaving messages, several interpretations are possible (see the discussion conducted in [Klo03]). The shown events usually may interleave with other events that are not explicitly mentioned in the diagram (e.g., in Figure 8 $!m1.?m1.!m4.?m4.!m2.?m2.!m3.?m3$ is a valid trace for both Grosu & Smolka and MSD). The difference is for the messages appearing on the diagram. For example, after receiving $m1$, are $m1$ or $m3$ allowed to appear? The *strict interpretation*, which is used in MSD, is that the diagram is complete with respect to occurrence specifications that are given in it explicitly. Therefore, neither $m1$ nor $m3$ are allowed right after an $m1$ message is matched. The *weak interpretation* (a form of which is used by Grosu & Smolka) is less restrictive with respect to the shown occurrence specifications. It only requires that the trace events occur in the specified order (e.g., an $m2$ message is in the future of $m1$) and may as well accept duplicates. Hence, the trace $!m1.?m1.!m3.?m3.!m2.?m2.!m3.?m3$ is valid for Grosu & Smolka, but not for MSD. Note that the trace $!m1.?m1.!m1.?m1.!m2.?m2.!m3.?m3$ is valid for both approaches; however, for MSD it is valid only if the first $m1$ message is considered as a prefix, and only the second $m1$ message is matched for the diagram.

Intuitively, interpretations with partial traces amount to filter out the behavior that is irrelevant to the categorization of traces: trace prefix, suffix, or extra interleaving events are ignored and categorization is based on the remaining part of the trace. It turns out that two operators of the UML 2.0 Sequence Diagrams, $ignore$ and its dual operator $consider$, allow us to further manipulate the set of events appearing in the traces. Not surprisingly, the decision of whether to work with partial or complete traces will have a strong impact on how these operators are interpreted. We will come back to this issue in Section 4.5.2 discussing $ignore$ and $consider$.



4.2 Introducing CombinedFragments

The OMG specification defines weak sequencing as the default composition operator for fragments. Accordingly, most semantics retain this operator to compose a CombinedFragment with the rest of the diagram (Störrle, STAIRS, Cengarle &

Knapp, Küster-Filipe, Knapp & Wuttke, Thread-Tag based, Template semantics).

Due to the weak sequencing, events that do not belong to the same lifeline can occur independently if they are not related by a path of messages. Figure 9 exemplifies this. Message $m1$ is located above the opt fragment, but there is actually no precedence relation: $m1$ can occur independently of messages $m2$ and $m3$. Similarly, placing something below a CombinedFragment does not necessarily mean that it comes after the messages inside the CombinedFragment. In Figure 9, there is an ordering constraint between $m2$ and $m3$ only because they share lifeline c. If the optional message $m2$ does not occur, then there is no constraint on $m3$. For example, trace $!m3.!m1.?m1.?m3$ is valid.

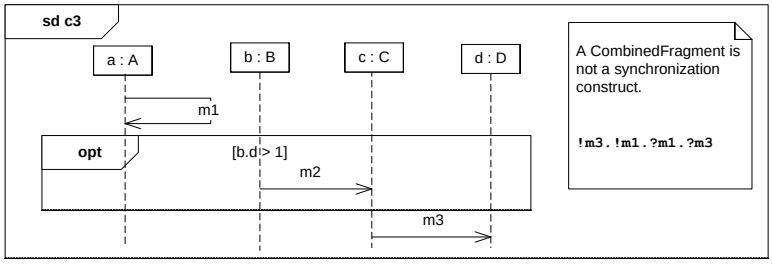


Figure 9: Composition with weak sequencing: above/below positions do not imply before/after relations

For such semantics, there is no synchronization point for crossing the borders of an operator. Technically, entering or exiting an operator is not an *Occurrence-Specification*. As far as we understand the OMG specification, the only OccurrenceSpecifications are (1) sending and receiving of Messages and (2) start and end of an ExecutionSpecification. These are the events that can appear in traces, and ordering constraints are defined for them only.

Also, the spatial extension of operators has no specific meaning if weak sequencing is used. In Figure 9, the opt box expands to all lifelines, but the meaning would be the same if it covered lifelines b and c only. This interpretation enjoys the property that an empty box is equivalent to no box (except for conformance-related operators, to be discussed in Section 4.5).

As a last example of how weak sequential composition determines the interpretation of diagrams, let us take an example with a loop (Figure 10). The meaning of the $loop$ operator is given as the recursive application of the seq operator. Because weak sequencing is used between the successive iterations of the loop, the trace where all the sending of $m1$ and $m2$ happens first, and all the receiving comes after it, is a valid trace.

While weak sequencing is the default according to the OMG specification, five semantics we reviewed introduce synchronization on entering and exiting fragments (Cavarra & Filipe, P-UMLaut, Hammal, MSD, CPN). This nonstandard interpretation is usually adopted for work using Sequence Diagrams for verification purposes. It is well known from previous work on MSCs that such graphical

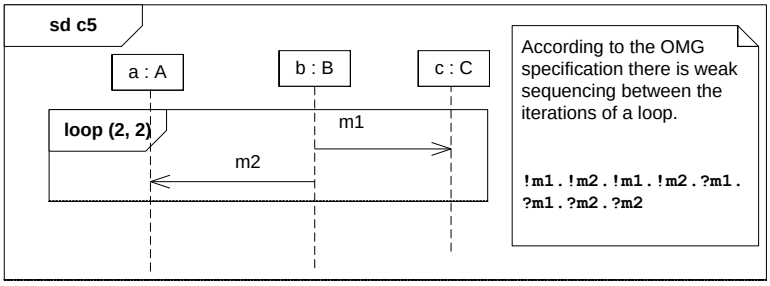


Figure 10: Loop means a weak sequencing between the iterations of the loop

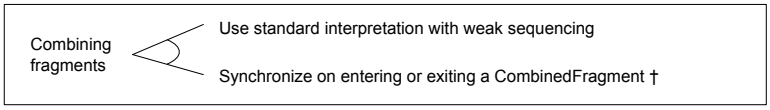
scenario languages are neither regular nor context-free, which raises decidability issues [MP05]. The synchronization then allows a reduction of the described partial orders of events, and makes properties easier to check. For example, assume the loop in Figure 9 can have an arbitrarily high number of iterations. The language is not regular with the standard interpretation, while it becomes regular if synchronization is enforced at each iteration.

In addition to reducing the expressive power of the language, the consequences of the synchronization are the following:

- Above/below positions now imply before/after relations, making the interpretation of the diagram close to the visual intuition;
- the spatial extension of boxes does matter, forcing each involved lifeline to synchronize;
- an empty box is no longer equivalent to no box; and
- the loop construct has an interpretation that is similar to the one of loops in programming languages.

None of the traces shown in Figure 9 and Figure 10 is valid for the semantics enforcing synchronization.

Some authors have proposed to retain the weak sequencing as the composition operator, except for loops where a new construct (*sloop*) makes it possible to consider strict sequencing of loop iterations [KW07].



4.3 Computing partial orders

The UML 2 specification defines the rules for computing the orderings between the OccurrenceSpecification on a simple diagram (see Section 2.2.1). This is usually a partial order because there can be independent events in the Interaction.

For example, in the leftmost diagram on Figure 11, $!m1$ and $!m2$ are not related while $?m1$ has to come before $?m2$.

With CombinedFragments this default ordering can be modified, e.g., in the middle diagram on Figure 11 the ordering between $?m1$ and $?m2$ is also relaxed, and we no longer have a complete ordering for events on the same Lifeline. An important thing to note is that when using par, the immediate predecessor and successor of OccurrenceSpecifications become sets. For example, in diagram $c7$ the predecessor of $!m3$ can be $!m1$ or $!m2$. Likewise, there is no such concept as the immediate next event; instead, there is a set of events. Finally, alternate fragments define several partial orders, one for each of their operands. In the rightmost diagram on Figure 11, there are two partial orders, one over the set of events $\{!m1, ?m1, !m3, ?m3\}$ and the other one over the set $\{!m2, ?m2, !m3, ?m3\}$.

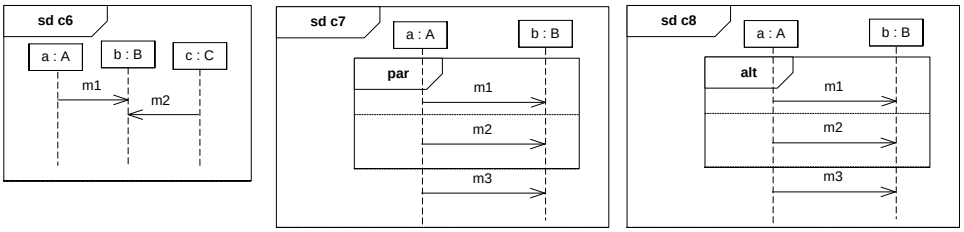


Figure 11: Partial orders in diagrams

When a diagram contains several CombinedFragments their effects combine. It may result in complex orderings, which are not trivial to calculate. Thus, a significant question about a semantics is how it computes the orderings for an Interaction.

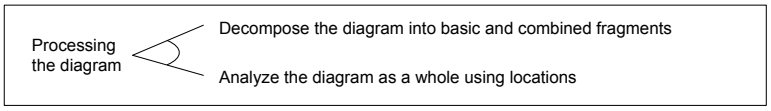
4.3.1 Processing the diagram

The approaches in the proposed semantics can be categorized in the following two main categories.

The semantics in the first category parse the diagram and decompose it. The CombinedFragments and the basic fragments in the diagram are identified (Störrle, P-UMLaut, Hammal, Thread-tag, CPN, Template semantics); some approaches even build a syntax tree from the elements of the diagram based on an abstract syntax (STAIRS, Cengarle & Knapp, Knapp & Wuttke). Usually, the parsing from a diagram's concrete syntax to this intermediate representation is not given in detail (some rules can be found in [Eic+05] or in [SVN08a] based on maximal independent sets). After the parsing, the semantics is computed by recursively unfolding the fragments and gluing them together based on rules defined for each of the operators.

The semantics in the second category analyze the diagram as a whole. The locations in the diagram are labeled, and the constraints about the relative ordering of locations are computed. The semantics connected to LSC use this ap-

proach (Cavarra & Filipe, Küster-Filipe, MSD). Küster-Filipe computes the event sequences leading to each of the locations. In MSD the first step is to obtain the valid cuts of the diagram from the analysis of locations.



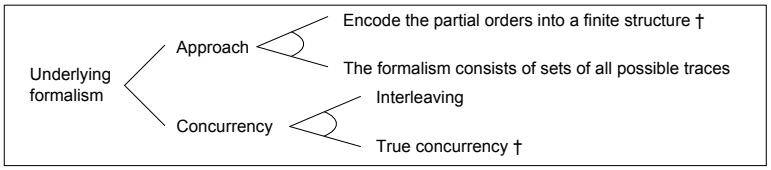
4.3.2 Underlying formalisms

In a semantics, the underlying formalism has a significant impact on how the orderings are computed and expressed. Table 5 summarizes the formalisms used in the surveyed approaches.

The diversity of formalisms in the approaches is the consequence of the *diversity of interests* for using Sequence Diagrams. Some authors define the semantics to check traces (e.g., Knapp & Wuttke), some to compute all possible traces of a diagram (e.g., Störrle), some use the semantics to support refinement-based development (e.g., STAIRS), or translate the diagrams into behavior models in order to connect to existing simulation or verification tools (e.g., P-UMLaut). The different purposes can be supported in either one or the other formalism more easily.

As a general comment, the underlying formalisms can be differentiated depending on whether they encode the partial orders into a finite structure (Cavarra & Filipe, Küster-Filipe, P-UMLaut, Grosu & Smolka, Hammal, MSD, Knapp & Wuttke, CPN, Template semantics), or they consist of sets of all possible traces (Störrle, STAIRS, Cengarle & Knapp, Thread-tag). With the first approach it is easier to verify traces, but it is usually feasible only with some syntactic restrictions (like Knapp & Wuttke allowing only basic fragments nested in a neg) and an interpretation that reduces the described partial orders (e.g., by synchronizing lifelines at the borders of fragments).

In MSD the model of not just one Interaction, but a system consisting of several Interactions is also defined. This is consistent with the fact that in MSD Interactions define only partial traces.



4.3.3 Choices and predicates

The definition of choices and predicates in the OMG specification is very permissive. As will be seen in this section, there are numerous options *what* and *when* to choose and *who* chooses.

Table 5: Underlying formalisms in the semantics

Name	Type of semantics	Concurrency
Störrle	Denotational semantics, rules for computing the set of traces	Interleaving
STAIRS	Denotational semantics, rules for computing the set of traces; operational semantics based on transitional systems	Interleaving
Cavarra & Filipe	Building Abstract State Machines, the ASMs accept or reject a trace	True concurrency
Cengarle & Knapp	Denotational semantics based on pomsets; operational semantics based on pomsets	True concurrency
Küster-Filipe	Denotational semantics based on event structures	True concurrency
P-UMLaut	Translating to M-nets	True concurrency
Grosu & Smolka	Translating to Büchi automaton, semantics is defined by the traces accepted by the automaton	Interleaving
Hammal	Denotational semantics based on graphs representing all traces	Interleaving
MSD	Building an alternating Büchi automaton, the automaton defines the trace-language accepted by the diagram	Interleaving
Knapp & Wuttke	Building an interaction automaton, the automaton observes traces and accepts or rejects them	Interleaving
Thread-tag based	Denotational semantics based on pomsets	True concurrency
CPN	Translating to Colored Petri nets	True concurrency
Template semantics	Operational semantics using Hierarchical Transition Systems	Interleaving

What An alt offers much more flexibility than an if construct in traditional programming languages would: several of its operands can have implicit true guards, from which one is non-deterministically chosen. Some approaches try to reduce this non-determinism. Cavarra & Filipe prescribe that the operands of the alt are evaluated from top to bottom, and the first one evaluated as true be chosen (a similar concept, *deterministic alt* was introduced in the UML 2 Testing Profile [OMG05]).

Who The UML 2 specification does not define *who* should make the choice between the operands of an alt. This can lead to *non-local choices*, a problem well studied in MSC [MGR05]. An example for non-local choice can be seen on Figure 12, where either instance a sends $m1$ or instance b sends $m2$, but not both. For semantics working with complete traces, non-local choices raise implementation problems: it may be impossible to implement a system, which shows the valid traces of the diagram. Most of the semantics

accept non-local choice as a consequence of having a high-level, powerful specification language.

When With the introduction of synchronization at the beginning and end of CombinedFragments (Section 4.2) some approaches specify a common point in time when all Lifelines have to make the choice.

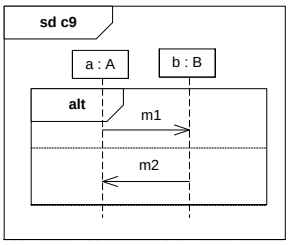


Figure 12: Simple non-local choice

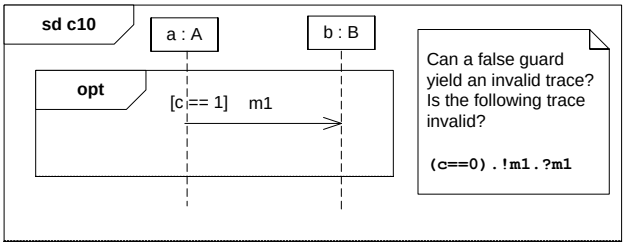


Figure 13: Handling of explicit guards

Thus handling choices is a complex issue. The main approaches used in the different semantics are the followings.

- *No explicit time point for the choice*: The sets of traces from each operand are computed independently and are combined with the rest of the diagram using the default weak sequencing to obtain all the possible traces of a diagram (Störrle, STAIRS, Cengarle & Knapp, Thread-tag).
- *Explicit time points for the choice on each Lifeline*: Lifelines process the diagram separately and choose between operands independently (Cavarra & Filipe, Küster-Filipe, Template semantics). Therefore, each Lifeline could make its choice at different times, but the semantics guarantees that all Lifelines choose the same operand (e.g., by fixing the evaluation order of operands).
- *Explicit global time point for the choice*: All involved lifelines synchronize before entering a choice, and only one global choice is made (P-UMLaut, Hammal, Grosu & Smolka, MSD, Knapp & Wuttke, CPN). These approaches typically use an automaton-based formalism, where one transition represents the taken choice for all Lifelines.

So far we only tackled implicit guards. Several semantics do not handle explicit guards. For the ones that do, a difference is how a false guard is interpreted. STAIRS processes guards similarly to constraints; thus a trace with a false guard is invalid, while for the other approaches (Cavarra & Filipe, Küster-Filipe P-UMLaut, Hammal, MSD, Knapp & Wuttke) a guarded choice cannot yield invalid traces. For example, the trace given in Figure 13 is invalid for STAIRS, while for the others it is not.

There are several options regarding who should evaluate the guard. The evaluation could be local to one Lifeline (STAIRS, Küster-Filipe), all Lifelines could interpret the guard separately (Cavarra & Filipe), or the guard could be evaluated globally (P-UMLaut, Hammal, MSD, Knapp & Wuttke). The latter option is consistent with an explicit global time point for the choice.

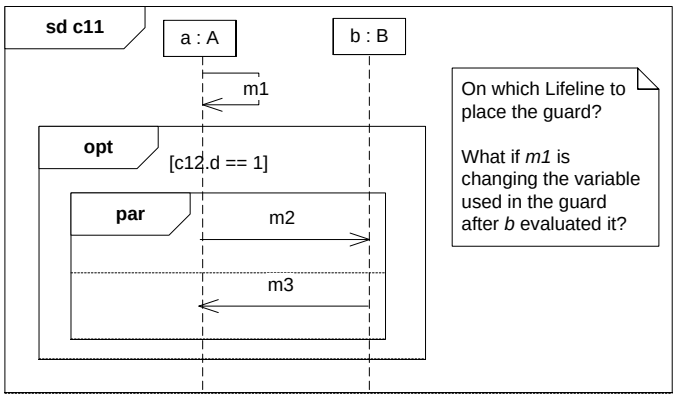
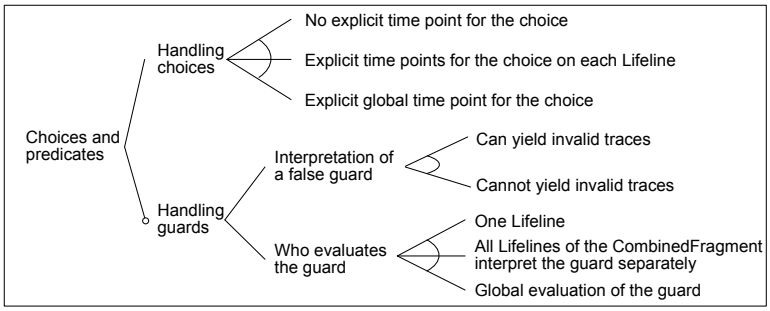


Figure 14: Data used in guards

Evaluating the guards separately or referring to global data may lead us to *scope* and *well-definedness* problems. As pointed out by Section 4.4 in [Eic+05] if Lifelines can evaluate the guards at different times, the value of the guard can change in the meantime. The UML 2 specification prescribes that the guard should be placed "on the lifeline where the first event occurrence will occur, positioned above that event, in the containing Interaction or InteractionOperand". However, as we mentioned before, "the" first event in an operand is not well defined, e.g., as in Figure 14.



4.4 Introducing Gates

As recalled in Figure 4, Gates allow Messages to go inside and outside of Interactions (*formalGate*), InteractionUses (*actualGate*) and CombinedFragments (*cfragmentGate*).

4.4.1 Gates on CombinedFragments

With the *cfragmentGate* type of Gate, messages can cross the boundaries of CombinedFragments (see Figure 14.9 of [OMG11]). Since *cfragmentGate* is allowed for any operator, it can yield problems.

As reported by Pickin in [Pic03], this will cause issues with loops. If a message goes into a loop, then it will have one sending end, but multiple receiving ones (see Figure 15). The *loop* operator is defined as a recursive application of the *seq* operator. Thus if the loop is unfolded, the result is a Message which has more than one receiving MessageEnds, which violates its constraints.

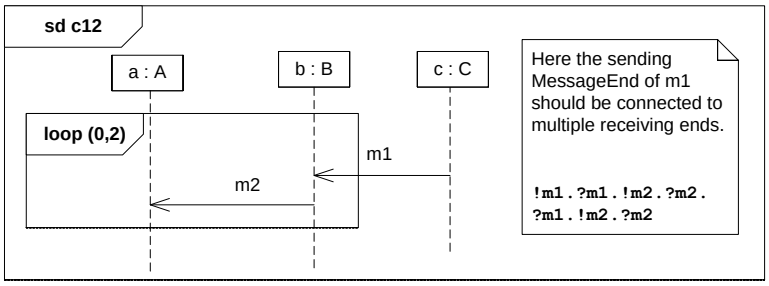


Figure 15: Message going into a loop

Most of the semantics do not consider *cfragmentGates*, or disallow it by their redefined abstract syntax (STAIRS, Cengarle & Knapp, Knapp & Wuttke). Our recommendation is also to remove it from the specification or heavily restrict its use, e.g., only to *critical* regions and *co-regions*.

4.4.2 Formal and actual Gates

The other two types of Gates (formal and actual) introduce a convenient facility for expressing complex scenarios: when a diagram includes a reference to another diagram (see Figure 16), Gates make it possible to model the passing of mes-sages. The referenced diagram has formalGates placed on its boundaries, allowing the representation of messages that come from, or go to, its environment. The environment is determined by the including diagram, where actualGates are placed at the borders of the *ref* box. Gates are MessageEnds that connect the Messages inside and outside the referenced diagram.

The surveyed semantics handle Gates in the following way. In STAIRS the set of Gates is defined as a subset of Lifelines, and events are defined when Gates receive or send Messages. Küster-Filipe adds symbolic events representing Gates, and extra orderings are added to the event structure accordingly. In P-UMLaut the referenced fragments are inlined before processing the Interaction.

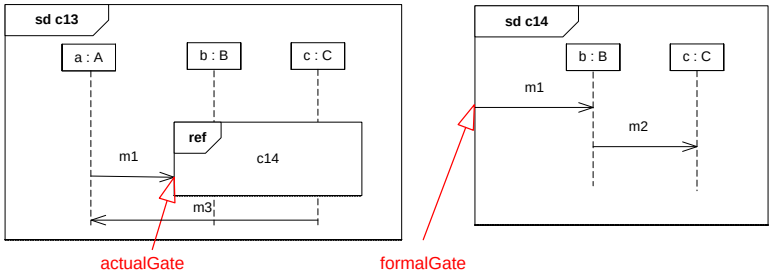
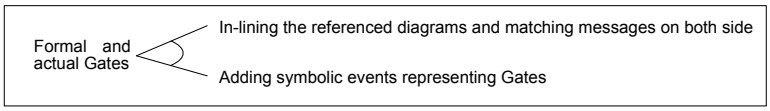


Figure 16: Formal and actual Gates



4.5 Interpretation of conformance-related operators

The interpretation of conformance-related operators (see our classification in Table 1) is a central issue in the definition of the semantics. Many papers about UML 2 Sequence Diagrams deal with this issue (or at least mention it). Indeed, quoting from [PJ04], "[*assert/negate/ignore/consider*] constructs open up a veritable pandora's box of expressions whose meaning is obscure." We provide here an overview of how the various semantics handle these constructs.

4.5.1 Assert and negate

The operators $assert$ and neg allow the specification of mandatory and forbidden behavior. Störrle was the first to discuss their interpretation in a formal semantics, and he identified several possible meanings for both operators [Stö04]. Further discussion of the neg construct can be found in [CK04b; RHS05a].

In practice, the chosen interpretation of $assert$ is consistent in all the semantics we reviewed. Let S be the fragment contained in an $assert$ box.

- The expression $assert(S)$ defines the same valid traces as S;
- Every trace that is not valid for S is invalid for $assert(S)$.

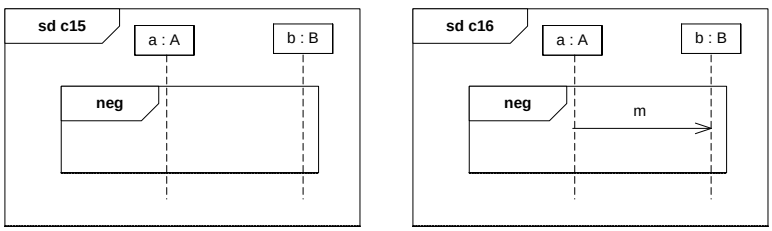


Figure 17: Negative fragments

Table 6: Interpretation of negative fragments on Figure 17 (Σ^* is the universe of traces)

	c15			c16		
Approach	Valid	Invalid	Inconclusive	Valid	Invalid	Inconclusive
Störrle	$\emptyset$	$\{\epsilon\}$	$\Sigma^* - \{\epsilon\}$	$\emptyset$	$\{!m.?m\}$	$\Sigma^* - \{!m.?m\}$
STAIRS	$\{\epsilon\}$	$\{\epsilon\}$	$\Sigma^* - \{\epsilon\}$	$\{\epsilon\}$	$\{!m.?m\}$	$\Sigma^* - \{\epsilon, !m.?m\}$
Cengarle & Knapp	$\{\epsilon\}$	$\emptyset$	$\Sigma^* - \{\epsilon\}$	$\{\epsilon\}$	$\{!m.?m\}$	$\Sigma^* - \{\epsilon, !m.?m\}$
Grosu & Smolka	$\Sigma^* - \{\epsilon\}$	$\{\epsilon\}$	$\emptyset$	$\Sigma^* - \{!m.?m\}$	$\{!m.?m\}$	$\emptyset$
Cavarra & Filipe, Küster-Filipe	$\emptyset$	Σ^*	$\emptyset$	$\emptyset$	$\{!m.?m\}$	$\Sigma^* - \{!m.?m\}$

This table focuses on the interpretation of negative fragments, and ignores diagram-wide issues that will be discussed in Section 4.5.3, such as whether an invalid prefix always makes an invalid trace. Hence, an invalid trace $!m.?m$ for the fragment may eventually yield a set of invalid traces $!m.?m.\Sigma^*$ for the diagram in Figure 17

Table 7: Interpretation of alternative negation operators (assuming each neg in Figure 17 is replaced by this operator)

	c15			c16		
Approach	Valid	Invalid	Inconclusive	Valid	Invalid	Inconclusive
Refuse	$\emptyset$	$\{\epsilon\}$	$\Sigma^* - \{\epsilon\}$	$\emptyset$	$\{!m.?m\}$	$\Sigma^* - \{!m.?m\}$
Not	$\Sigma^* - \{\epsilon\}$	$\{\epsilon\}$	$\emptyset$	$\Sigma^* - \{!m.?m\}$	$\{!m.?m\}$	$\emptyset$

Operator $refuse(S)$ from STAIRS: all valid and invalid traces of S are invalid, there is no valid trace. Operator $not(S)$ from Cengarle & Knapp and Knapp & Wuttke: anything but S, there is no inconclusive trace.

The neg construct is more controversial, and several interpretations have been adopted. Table 6 illustrates some of the differences between them. They may be described as follows:

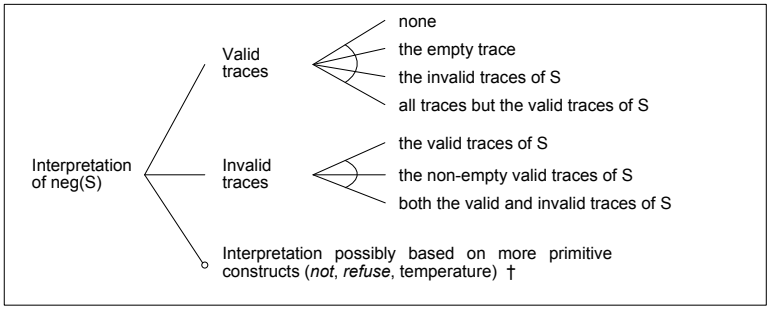
- For Störrle, the preferred interpretation is that $neg(S)$ flips the valid and invalid traces of S. Inconclusive traces are left unchanged. This interpretation enjoys the property that $neg \circ neg = Id$.
- In STAIRS, the empty trace is the only valid trace of $neg(S)$. Both valid and invalid traces of S are invalid for $neg(S)$. This interpretation ensures that neg is monotonic with respect to the refinement relation chosen by the authors: if S is refined by S', then $neg(S)$ is refined by $neg(S')$. It is worth noting that neg is not primitive for this semantics. It is interpreted as a choice between $skip$ and $refuse(S)$, where $refuse$ is the primitive concept. The meaning of $refuse$ is shown in Table 7.

- For Cengarle & Knapp, the empty trace is also the only valid trace of $neg(S)$. Non-empty traces that are valid for S are invalid for $neg(S)$. All other traces are inconclusive. This interpretation enjoys the property that $neg(skip) = skip$, that is, an empty neg box is equivalent to no box. Like in STAIRS, monotonicity with respect to refinement is considered, but with a different notion of refinement (and a different monotonicity property). Also, the semantics introduces a new operator, not, that is more primitive than neg and upon which the meaning of neg is built (see Table 7).

- For Grosu & Smolka, $neg(S)$ is interpreted as "anything but S"[2]. All traces, but the valid traces of S, are valid for $neg(S)$. This interpretation is relevant to verification purposes when the aim is to check that a system never exhibits the forbidden behavior.

- In Küster-Filipe, Cavarra & Filipe and MSD, $neg(S)$ is syntactic sugar for a global false predicate put at the end of S. Table 6 shows the interpretation of Küster-Filipe and Cavarra & Filipe. Note that the MSD interpretation would be different because the diagrams would describe partial traces (see Section 4.1.3). These three semantics are actually specific in their expression of mandatory and forbidden behavior because they inherit from modalities previously defined for LSCs. Inside a diagram, individual locations are assigned a hot (mandatory) or cold (possible) temperature. The operator $assert$ is syntactic sugar to turn all inside locations to hot, and neg adds a (hot) false predicate. In our interpretation of Figure 17, we assumed that the locations inside the neg have a cold temperature (otherwise, in the righthand diagram, the system would be required to exhibit $?m.!m$ and reach the false predicate, so that all traces would be invalid as in the empty neg).

To sum up, all semantics agree that $neg(S)$ should turn valid traces of S to invalid. However, there are differences in the way invalid traces of S are handled. Also, the empty trace is sometimes assigned a specific treatment.



[2]The neg is thus interpreted like the not operator mentioned just above.

4.5.2 Ignore and consider

The operators $ignore$ and $consider$ affect the notion of conformance to a diagram, by changing the alphabet from which the valid and invalid traces are built. They make it possible to account for the sending and receiving of messages not explicitly represented in the diagram. The description of these operators is unclear in the OMG specification, and few semantics address them. For the ones that do, the proposed interpretation depends on whether the semantics works with complete or partial traces.

Störrle, Cengarle & Knapp and Knapp & Wuttke fall in the first category, using an interpretation with complete traces. For them, by default, a valid trace can only contain OccurrenceSpecifications shown in the diagram. The operators $ignore$ and $consider$ both allow the extension of traces with additional OccurrenceSpecifications. Ignoring a message m means that occurrences of $?m$ and $!m$ may interleave with the explicitly specified behavior. Assume that S is a basic interaction fragment involving a single message n and having one valid trace $!n.?n$. Then, $ignore(\{m\}, S)$ means that all traces of the form $(?m|!m)^*.!n.(?m|!m)^*.?n.(?m|!m)^*$ are valid. Note that the set of ignored messages could contain n, in which case we would accept traces with multiple occurrences of n. The dual operator $consider(\{m\}, S)$ is interpreted as $ignore(M - \{m\}, S)$, where M is the set of all possible messages. Its intuitive meaning is thus "ignore everything but m".

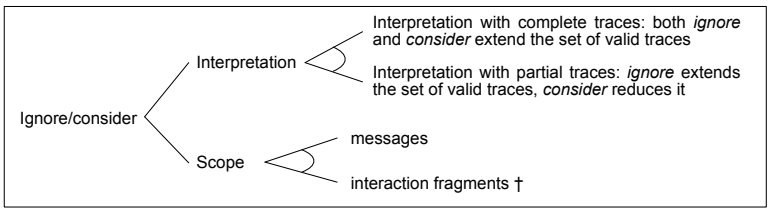
Semantics considering partial traces, like MSD, cannot have the same interpretation. Messages not shown in a diagram are already "ignored" by default, and a system trace may contain an arbitrary prefix (resp. a suffix) before (resp. after) the shown behavior occurs. The ignore operator is useful only in the case where we want to allow multiple occurrences of the shown messages, in the temporal window of the shown behavior. If interaction S involves message n and does not involve message m, then

- $ignore(\{m\}, S)$ is equivalent to S;
- $ignore(\{n\}, S)$ has a larger set of valid traces than S.

As regards the $consider$ operator, its interpretation departs from the one given by semantics that work with complete traces. In MSD, $consider$ is a means to *reduce* the set of valid traces. It is useful when the "considered" messages would be ignored by default. Hence, if interaction S does not involve message m, $consider(\{m\}, S)$ results in a more restrictive interaction than S: it no longer allows occurrences of m in the temporal window of S.

MSD exhibits additional specificities in the way $ignore$ and $consider$ are handled. First, the operators are changed to specify interaction fragments (not just messages) to be ignored or considered. For example, we may "consider" a fragment consisting of message m_1 sent by lifeline l_a to lifeline l_b, followed by message m_2 from l_b to l_a. This increases the expressiveness compared with just con-

sidering m_1 and m_2. Second, the introduced fragments are assigned a temperature, offering an opportunity to distinguish cold and hot violations when the "considered" behavior inopportunely occurs.



4.5.3 Conformance-related operators in complex diagrams

Up to now, we discussed the interpretation of each conformance-related operator taken in isolation. But in complex diagrams, conformance-related operators can include, or be nested into, other constructs.

Let us consider the nesting of conformance-related operators into each other. This raises issues such as the interpretation of multiple assertions, multiple negations, assertions of negations, negation of fragments with considered and ignored messages, and messages that are both ignored and considered. Of course, the semantics dealing with these constructs will assign a precise meaning to such cases. However, the assigned meaning may defeat intuition, with the risk of producing diagrams that users do not properly understand. To illustrate this point, double negation is a good example (see Figure 18). It is striking that the various semantics offer all possibilities for the categorization of trace $!m.?m$:

- The trace is valid for Störrle;
- It is invalid for STAIRS and all interpretations adding a false global predicate (MSD, Küster-Filipe, Cavarra & Filipe);
- It is inconclusive for Cengarle & Knapp.

Whatever the chosen semantics, care must be taken that it really captures the meaning intended by the specifier.

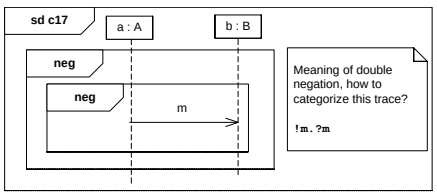


Figure 18: Nesting conformance-related operators

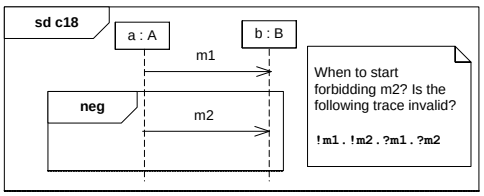


Figure 19: Borders of conformance-related operators

Syntactic restrictions may reduce the risk of counterintuitive interpretations. In [Stö04], Störrle came to the conclusion that neg should not be used as an ordinary operator. It should be used only at the top level of a diagram, to indicate that the diagram describes a forbidden scenario. This avoids intriguing cases where neg is nested into other operators. Other authors (Knapp & Wuttke, Grosu & Smolka) forbid the nesting of operators into a negation, so that negated fragments can only contain basic interactions. Their motivation, however, is not to preserve intuition but to ease verification of conformance: they want the detection of invalid traces to be kept decidable. As a general rule, one may put syntactic restrictions on conformance-related operators for either purpose, for keeping diagrams intuitive or for facilitating their usage in verification activities.

Apart from the case where the operator is used only at the top level (as recommended by Störrle for the neg operator), an important semantic issue is how conformance-related operators are combined with the rest of the diagram. The general decision on whether to synchronize or not on the borders of boxes has an impact on the categorization of traces. In Figure 19, the shown trace is not invalid if synchronization is enforced. This raises the issue of when to start requiring ($assert$), accepting ($ignore$) or forbidding (neg, and also $consider$ in its interpretation using partial traces) the communication events appearing in the operator.

Another issue is how to interpret sequencing (whether weak or strong) when the prefix of a trace completely traverses a negative region. Figure 20 exemplifies this issue. Will a trace starting with prefix $!m1.?m1.!m2.?m2$ be categorized as invalid whatever the suffix? Almost all the semantics answer by yes. This has the advantage of facilitating the identification of invalid traces: decision can be taken locally, independently of what will happen subsequently (see the discussion conducted in [CK04b]). However, a different interpretation is chosen in STAIRS: the trace is inconclusive if the suffix does not match. Note that Figure 20 involves a neg, but a similar example could be built with a trace prefix violating an $assert$.

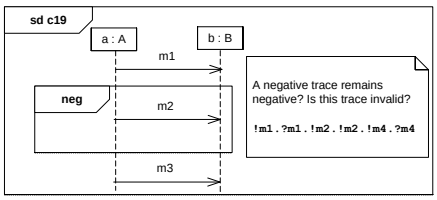


Figure 20: Negative trace remains negative?

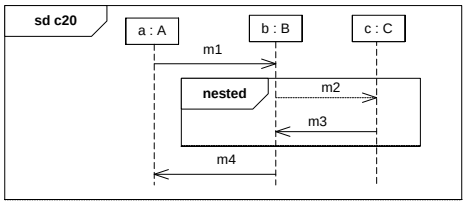
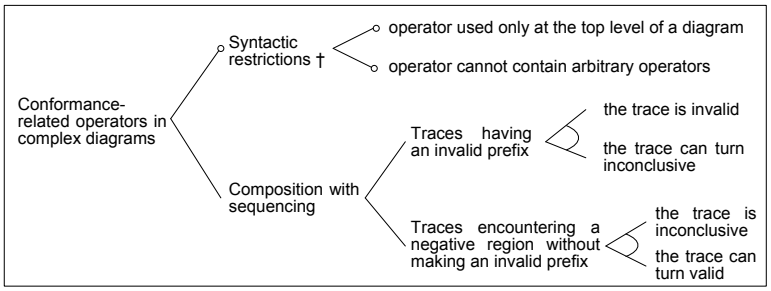


Figure 21: Nested operator in MSD

What about then the continuation of traces that do not completely traverse a negative region? For example, in Figure 20, is the trace $!m1.?m1.!m2.!m3.?m3$ valid or inconclusive? In the semantics of Grosu & Smolka, the trace would be valid. For Cengarle & Knapp and STAIRS, the only valid traces would be $!m1.?m1.!m3.?m3$ and $!m1.!m3.?m1.?m3$. For semantics adding a global false

predicate (MSD, Küster-Filipe, Cavarra & Filipe), there can be no valid traces: only invalid or inconclusive ones.

Technically, for the latter three semantics, completely traversing the *neg* yields a hot violation, while failing to traverse the *neg* is a cold violation. In both cases, the local violation determines the categorization of the overall trace (invalid, inconclusive) whatever the suffix. Hence, the scope of the local violation is actually global. The authors of MSD have proposed a new operator, *nested*, allowing them to restrict the scope of cold violations. It is then possible to have valid continuations of traces that do not completely traverse a fragment. Let us recall that in MSD, the primitive concept to express mandatory/forbidden behavior is the temperature. In Figure 21, message *m2* is cold (indicated by a dotted line) while message *m3* is hot (indicated by a solid line). It means that *m2* may occur, and if it does, then *m3* is required. Since MSD only considers synchronous messages, we do not distinguish the sending and receiving of messages. In a nested fragment, a cold violation is confined: the trace continues in the enclosing fragment. In Figure 21, the trace *m1.m4* is indeed valid. It would be inconclusive if *m2* and *m3* were in the plain fragment.



4.5.4 Traces being both valid and invalid

Ambiguous diagram can be constructed, where a given trace is both valid and invalid. In the example of Figure 22, this is due to non-determinism. A given event in the trace can be considered as occurring either inside or outside the scope of the conformance-related operator, depending on some non-deterministic choice. Parallel constructs come with similar ambiguities.

It may seem that the problem comes with the nesting of conformance-related operators into non-deterministic constructs. But the example of Figure 23, borrowed from [CK04b], only involves weak sequencing. Intuitively, it is not clear whether the occurrence of *m1* in the trace falls into the scope of *neg*, or may be considered as posterior to the *neg*. Note that the interpretations given by the various semantics are not necessarily ambiguous. For semantics adding a false predicate at the end of the *neg* fragment, trace $!m1.?m1$ is clearly invalid. For STAIRS, it is valid (only a double occurrence of *m1* would be invalid). However, for Cengarle & Knapp, the trace is both valid and invalid.

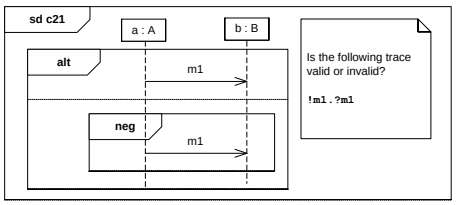


Figure 22: Ambiguity due to non-determinism

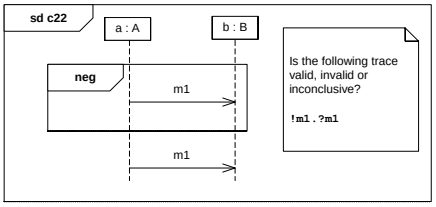


Figure 23: Ambiguous scope of a conformance-related operator

Figure 24 illustrates yet another possibility for introducing ambiguity. Here, the ambiguous case comes from the consideration for the values of message parameters, in a diagram where the two occurrences of message m cannot be distinguished. The shown trace may be categorized as valid or invalid, depending on whether the final assertion is $2 > 1$ or $1 > 2$.

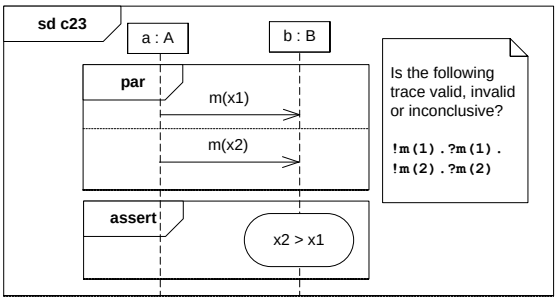
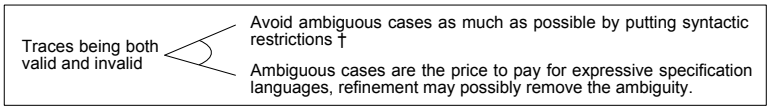


Figure 24: Ambiguous cases due to the consideration of data values

These various examples show that it is extremely difficult to get rid of ambiguous cases. We may put syntactic restrictions to avoid some of the cases (e.g., use only a deterministic if-then-else form of alt constructs, or use neg only at the top level of the diagram), but avoiding all of them by construction would probably require us to sacrifice too much in terms of language expressiveness. Indeed, from our analysis of work dealing with conformance-related operators, all surveyed approaches face cases where a trace can be both valid and invalid. In general, checking whether a diagram is ambiguous is an undecidable problem. From previous work on MSCs [MP05], we know that language complementation and intersection are not decidable for graphical scenarios. Then, if ambiguous cases are to be avoided, it is probably wise not only to put syntactic restrictions, but also to adopt an interpretation that brings the semantics of diagrams to a regular language (synchronize on entering and exiting fragments, encode the partial orders into an automaton). In this way, the ambiguous cases not covered by the syntactic restrictions can be detected and reported to the user.

Not all authors explicitly mention the existence of ambiguous diagrams. For the ones who do [CK04b; LS06; Stö04], this is not necessarily considered as a

problem. In [CK04b], ambiguous diagrams are called *overspecified* interactions. There may exist refinements that remove the ambiguity, so that an overspecified interaction may indeed have an implementation. For example, both Figure 22 and Figure 23 are overspecified interactions according to Cengarle & Knapp. Figure 23 is not implementable, but Figure 22 is implementable by the *skip* process.



5 Illustration on using the collected semantic choices

To illustrate how the collection of choices in Section 4 could help to develop a new language based on Sequence Diagrams, we present our experience on creating *TEst Requirement language for MObile Setting* (TERMOS) [Wae+10]. Mobile settings introduce new challenges not present in traditional distributed systems, including high dynamicity of the system topology or communication with unknown partners in local vicinity. Existing scenario languages do not offer concepts to account for the dynamically changing structure and context, nor do they offer constructs to represent broadcast communication in local vicinity. We proposed to extend Sequence Diagrams to fill these gaps with a separate spatial view to depict topology changes, stereotypes to express broadcast, and configurations, etc. These extensions were built into TERMOS, a language for expressing test requirements for mobile systems. When defining the semantics of the new language, assigning a meaning to these extensions was relatively straightforward. The core UML elements were the ones presenting several choices.

The main goal of the language is to check whether an execution trace satisfies a given scenario. In order to fulfill this purpose, the choices were handled according to the following design decisions (Table 8 summarizes them).

Interpretation of a basic Interaction: Because real execution traces should be checked, it should be made possi- ble to exactly identify the sending and receiving events of all messages; thus the definition of a trace contains unique IDs. Not all traces are relevant for a requirement; hence the trace universe is partitioned into three classes (valid, invalid and inconclusive traces). A test requirement scenario should be definitely a partial description because it captures just a fragment of the system's behavior. We wanted a very flexible language; thus in TERMOS both a prefix and suffix are allowed, messages not depicted on the diagram can interleave, and the weak interpretation is used.

Introducing CombinedFragments: In order to make the checking of a trace easier, entering and exiting a CombinedFragment is treated as a synchronization point.

Table 8: Summary of choices taken in TERMOS

Interpretation of a basic Interaction	What is a trace?	Definition of a trace contains unique message Ids
	Categorizing traces	Valid, invalid, inconclusive
	Complete or partial traces	Partial traces (prefix/suffix allowed, extra messages can interleave, weak interpretation for duplicates)
Introducing CombinedFragments	Combining fragments	Synchronization on entering or exiting a CombinedFragment
Computing partial orders	Processing the diagram	Process the diagram as a whole using locations
	Underlying formalisms	Interleaving semantics, encode the partial orders into a finite automaton
	Choices and predicates	Explicit global time point for the choice, a false guard does not yield an invalid trace, guards are evaluated globally
Introducing Gates	Gates on Combined-Fragments Formal and actual Gates	Gates were disallowed completely
Interpretation of conformance-related operators	Assert and negate	Instead of *neg* as an operator, a global false predicate can be put at the end of the diagram
	Ignore and consider	Using partial traces and the weak interpretation makes *ignore* redundant, *consider* reduces the set of valid traces
	Conformance-related operators in complex diagrams	Nesting is restricted, traces having an invalid prefix are invalid
	Traces being both valid and invalid	Syntactic restrictions avoid some of the ambiguous cases

Computing partial orders: The defined formal semantics was inspired by LSC's semantics, as the goals of TERMOS are similar to LSC (like expressing requirements or depicting partial traces). Therefore, TERMOS uses also the location concept and processes the diagram as a whole. A state-based formalism was chosen because it makes checking of a given trace feasible. An automaton is built for the whole diagram, which represents all instances. Choices are made globally and there is a global time point for the choice to ease the verification of traces. The time point is materialized by a state with outgoing transitions for the alternatives. If the choice is guarded, the explicit guards appear as transition labels. Their evaluation is thus made globally from the state. A false guard does not yield an invalid trace and the trace becomes inconclusive.

Introducing gates: Requirement scenarios should be kept simple; thus Gates were disallowed.

Interpretation of conformance-related operators: The options for conformance-related operators were chosen in order to make checking a trace feasible. Handling the neg operator combined the approaches from MSD (using a global false predicate instead of neg as an operator) and Störrle (negation is for the whole Interaction, it can appear only at the end of the top-level fragment). Because of partial traces and weak interpretation the ignore operator is not needed, consider reduces the set of valid traces. To ease the detection of valid traces the nesting of operators is heavily restricted. We want to avoid ambiguous cases as much as possible; hence again some syntactic restrictions. Furthermore, we defined checks on the generated automaton, to detect some remaining cases of non-deterministic categorization into valid and invalid traces.

The collection of the semantic choices helped us to identify what should be decided for the new language. As can be seen from this case study, it provides a structured framework to consider the various options and to make design decisions that suit the purpose of the newly defined language.

6 Conclusion

Due to the variety of usage for scenarios, there is nothing such as "the" semantics of UML 2 Sequence Diagrams. The OMG group has always insisted on the fact that the standard enables specialization of parts of UML for a particular situation or domain. As regards Sequence Diagrams, it would probably be an impossible task to exhibit an "all-in-one" semantics fitting purposes as diverse as the description of example interactions, of test cases, or of checkable properties.

Flexibility to assign different interpretations to diagrams leaves UML practitioners with a difficult problem: the one of selecting a semantics well suited for their purpose. There is a lack of a clear picture of available options. Since the pioneering work of Störrle for giving a formal meaning to UML 2 Sequence Diagrams, a number of alternative semantics have flourished, and the research presented in this chapter is an attempt to gain a synthetic view of the choices that underlie

them.

Our approach was to select a sample of 13 semantics and to systematically identify the points in which they differ. We took care to include widely referenced work, as well as less-referenced one that may be representative of more specialized concerns. We created a categorization of choices, ranging from the interpretation of basic diagrams to the interpretation of advanced constructs such as conformance-related operators. For each choice, we listed the options encountered in the analyzed sample. Our discussion of options tried to be very practical, by showing their concrete consequence on examples of diagrams.

We ended up with a structured representation of the various choices and options, inspired by feature models. The proposed categorization already provided useful guidance for designing the semantics of a real-life language, TERMOS. We hope that it will be helpful as well to other UML practitioners searching for a suitable semantics or wanting to define a new semantics in their own problem domain.

Our work may admittedly need future extensions, to account for the large number of semantics existing today or the ones that will continue to emerge in the next years. We believe that the feature-model-like representation offers a convenient support for documenting the semantic variants, and for updating the existing categorization. In the long run, it could be imagined that the OMG standard for Sequence Diagrams includes a model similar to ours, so as to make the semantic variation points more explicit. If such is the case, we would recommend that some of the options marked as non-standard (like working with partial traces, considering a categorization into valid/other or invalid/other, or synchronizing on the borders of fragments) be explicitly mentioned in the specification. We repeatedly found them in several of the surveyed semantics, and feel that they are very useful to address some recurring needs related to verification and testing purposes.

Acknowledgments We would like to thank the anonymous reviewers for their helpful comments and suggestions. We really appreciated their detailed feedback.

References

[Bow06] J. K. F. Bowles. “Decomposing Interactions”. In: *11th International Conference on Algebraic Methodology and Software Technology (AMAST 2006)*. 2006, pp. 189–203. DOI: 10.1007/11784180_16.

[Bro+08] M. Broy, M. V. Cengarle, H. Grönniger, and B. Rumpe. *Modular description of a comprehensive semantics model for the UML (Version 2.0)*. Tech. rep. 2008-06. Carl-Friedrich-Gaus-Fakultat, Technische Universitat Braunschweig, 2008.

[Cen07] M. V. Cengarle. “System model for UML – The interactions case”. In: *Methods for Modelling Software Systems (MMOSS)*. Dagstuhl Seminar Proceedings 06351. 2007. URL: http://drops.dagstuhl.de/opus/volltexte/2007/857.

[CGW06] M. V. Cengarle, P. Graubmann, and S. Wagner. “Semantics of UML 2.0 Interactions with Variabilities”. In: *Electr. Notes Theor. Comput. Sci.* 160 (2006), pp. 141–155.

[CK04a] A. Cavarra and J. Küster-Filipe. “Formalizing Liveness-Enriched Sequence Diagrams Using ASMs”. In: *Abstract State Machines 2004. Advances in Theory and Practice*. Vol. 3052. LNCS. 2004, pp. 62–77. DOI: 10.1007/978-3-540-24773-9_6.

[CK04b] M. V. Cengarle and A. Knapp. “UML 2.0 Interactions: Semantics and Refinement”. In: *3rd Int Workshop on Critical Systems Development with UML (CSDUML04, Proceedings), Technical Report TUM-I0415*. 2004, pp. 85–99.

[CK05a] A. Cavarra and J. Küster-Filipe. “Combining Sequence Diagrams and OCL for Liveness”. In: *Electronic Notes in Theoretical Computer Science* 115 (2005), pp. 19–38. DOI: 10.1016/j.entcs.2004.09.025.

[CK05b] M. V. Cengarle and A. Knapp. *Operational Semantics of UML 2.0 Interactions*. Tech. rep. TUM-I0505. Institut für Informatik, Technische Universitat München, 2005.

[CK08] M. V. Cengarle and A. Knapp. *An Institution for UML 2.0 Interactions*. Tech. rep. TUM-I0808. Technische Universität München, 2008. URL: http://www4.in.tum.de/~cengarle/papers/TUM-I0808.pdf.

[DH01] W. Damm and D. Harel. “LSCs: Breathing Life into Message Sequence Charts”. In: *Formal Methods in System Design* 19.1 (2001), pp. 45–80. DOI: 10.1023/A:1011227529550.

[DHC07] H. Dan, R. M. Hierons, and S. Counsell. “A Thread-tag Based Semantics for Sequence Diagrams”. In: *Proceedings of the Fifth IEEE International Conference on Software Engineering and Formal Methods*. 2007, pp. 173–182. DOI: 10.1109/SEFM.2007.3.

[Eic+05] C. Eichner, H. Fleischhack, R. Meyer, U. Schrimpf, and C. Stehno. “Compositional Semantics for UML 2.0 Sequence Diagrams Using Petri Nets”. In: *SDL 2005: Model Driven Systems Design*. 2005, pp. 133–148. DOI: 10.1007/11506843_9.

[Fer+07] J. M. Fernandes, S. Tjell, J. B. Jorgensen, and O. Ribeiro. “Designing Tool Support for Translating Use Cases and UML 2.0 Sequence Diagrams into a Coloured Petri Net”. In: *Proceedings of the Sixth International Workshop on Scenarios and State Machines*. 2007, p. 2. DOI: 10.1109/SCESM.2007.1.

[GS05] R. Grosu and S. A. Smolka. “Safety-Liveness Semantics for UML 2.0 Sequence Diagrams”. In: *Proceedings of the Fifth International Conference on Application of Concurrency to System Design*. 2005, pp. 6–14. DOI: 10.1109/ACSD.2005.31.

[Hal+06] H. H. Hallal, S. Boroday, A. Petrenko, and A. Ulrich. “A formal approach to property testing in causally consistent distributed traces”. In: *Form. Asp. Comput.* 18 (1 2006), pp. 63–83. DOI: 10.1007/s00165-005-0082-9.

[Ham06] Y. Hammal. “Branching Time Semantics for UML 2.0 Sequence Diagrams”. In: *Formal Techniques for Networked and Distributed Systems - FORTE 2006*. Vol. 4229. LNCS. 2006, pp. 259–274. DOI: 10.1007/11888116_20.

[Hau+05] Ø. Haugen, K. E. Husa, R. K. Runde, and K. Stølen. “Why Timed Sequence Diagrams Require Three-Event Semantics”. In: *Scenarios: Models, Transformations and Tools*. 2005, pp. 1–25. DOI: 10.1007/11495628_1.

[HKM07] D. Harel, A. Kleinbort, and S. Maoz. “S2A: A Compiler for Multimodal UML Sequence Diagrams”. In: *10th International Conference on Fundamental Approaches to Software Engineering (FASE 2007)*. 2007, pp. 121–124. DOI: 10.1007/978-3-540-71289-3_11.

[HM03] D. Harel and R. Marelly. *Come, Let’s Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer-Verlag, 2003.

[HM08] D. Harel and S. Maoz. “Assert and negate revisited: Modal semantics for UML sequence diagrams”. In: *Software and Systems Modeling* 7.2 (2008), pp. 237–252. DOI: 10.1007/s10270-007-0054-z.

[HS03] Ø. Haugen and K. Stølen. “STAIRS - Steps To Analyze Interactions with Refinement Semantics”. In: *The Unified Modeling Language. Modeling Languages and Applications*. Vol. 2863. LNCS. 2003, pp. 388–402. DOI: 10.1007/978-3-540-45221-8_33.

[ITU11] International Telecommunication Union. *Message Sequence Chart (MSC)*. Recommendation Z.120. 2011. URL: http://www.itu.int/rec/T-REC-Z.120.

[Kan+90] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Tech. rep. CMU/SEI-90-TR-021. Carnegie-Mellon University Software Engineering Institute, 1990.

[Klo03] J. Klose. “Live Sequence Charts: A Graphical Formalism for the Specification of Communication Behavior”. PhD thesis. Carl von Ossietzky Universitat Oldenburg, 2003.

[Küs06] J. Küster-Filipe. “Modelling Concurrent Interactions”. In: *Theoretical Computer Science* 351.2 (2006), pp. 203–220. DOI: 10.1016/j.tcs.2005.09.068.

[KW07] A. Knapp and J. Wuttke. “Model Checking of UML 2.0 Interactions”. In: *Models in Software Engineering*. Vol. 4364. LNCS. 2007, pp. 42–51. DOI: 10.1007/978-3-540-69489-2_6.

[LS06] M. Lund and K. Stølen. “A Fully General Operational Semantics for UML 2.0 Sequence Diagrams with Potential and Mandatory Choice”. In: *FM 2006: Formal Methods*. 2006, pp. 380–395. DOI: 10.1007/11813040_26.

[Lun08] M. S. Lund. “Operational analysis of sequence diagram specifications”. PhD thesis. University of Oslo, 2008.

[MGR05] A. J. Mooij, N. Goga, and J. M. Romijn. “Non-local Choice and Beyond: Intricacies of MSC Choice Nodes”. In: *Fundamental Approaches to Software Engineering*. 2005, pp. 273–288. DOI: 10.1007/978-3-540-31984-9_21.

[MP05] A. Muscholl and D. Peled. “Deciding Properties of Message Sequence Charts”. In: *Scenarios: Models, Transformations and Tools*. 2005, pp. 43–65. DOI: 10.1007/11495628_3.

[MW08] Z. Micskei and H. Waeselynck. *A survey of UML 2.0 sequence diagrams’ semantics*. Tech. rep. 08389. Laboratoire d’Analyse et d’Architecture des Systemes (LAAS), 2008, pp. 1–37.

[OMG05] Object Management Group. *UML Testing Profile v1.0 (U2TP)*. 2005. URL: http://www.omg.org/technology/documents/formal/test_profile.htm.

[OMG11] Object Management Group. *Unified Modeling Language (UML) 2.4.1 Superstructure Specification*. formal/2011-08-06. 2011.

[Pic03] S. Pickin. “Test des composants logiciels pour les télécommunications”. PhD thesis. Université de Rennes, 2003.

[PJ04] S. Pickin and J.-M. Jézéquel. “Using UML Sequence Diagrams as the Basis for a Formal Test Description Language”. In: *Integrated Formal Methods*. Vol. 2999. LNCS. 2004, pp. 481–500. DOI: 10.1007/978-3-540-24756-2_26.

[RHS05a] R. K. Runde, Ø. Haugen, and K. Stølen. “How to transform UML neg into a useful construct”. In: *Norsk Informatikkonferanse (NIK’05)*. 2005, pp. 55–66.

[RHS05b] R. K. Runde, Ø. Haugen, and K. Stølen. “Refining UML interactions with underspecification and nondeterminism”. In: *Nordic J. of Computing* 12.2 (2005), pp. 157–188.

[Run07] R. Runde. “STAIRS – understanding and developing specifications expressed as UML interaction diagrams”. PhD thesis. University of Oslo, 2007.

[SC06] B. Sengupta and R. Cleaveland. “Triggered Message Sequence Charts”. In: *Transactions on Software Engineering* 32.8 (2006), pp. 587–607. DOI: 10.1109/TSE.2006.82.

[Sel04] B. Selic. “On the Semantic Foundations of Standard UML 2.0”. In: *Formal Methods for the Design of Real-Time Systems*. Vol. 3185. LNCS. 2004, pp. 75–76. DOI: 10.1007/978-3-540-30080-9_6.

[Stö03a] H. Störrle. “Assert, Negate and Refinement in UML-2 Interactions”. In: *Workshop on Critical Systems Development with UML (CSDUML'03), Technical Report TUM-I0317*. 2003, pp. 79–94.

[Stö03b] H. Störrle. “Semantics of Interactions in UML 2.0”. In: *IEEE Symposium on Human Centric Computing Languages and Environments*. 2003, pp. 129–136. DOI: 10.1109/HCC.2003.1260216.

[Stö04] H. Störrle. *Trace Semantics of Interactions in UML 2.0*. Tech. rep. Institut für Informatik, Ludwig-Maximilians-Universität München, 2004.

[SVN08a] H. Shen, A. Virani, and J. Niu. “Formalize UML 2 Sequence Diagrams”. In: *High Assurance Systems Engineering Symposium, 2008. HASE 2008. 11th IEEE*. 2008, pp. 437–440. DOI: 10.1109/HASE.2008.51.

[SVN08b] H. Shen, A. Virani, and J. Niu. *Formalize UML 2 Sequence Diagrams*. Tech. rep. CS-TR-2008-13. University of Texas at San Antonio, 2008.

[Wae+10] H. Waeselynck, Z. Micskei, N. Rivière, Á. Hamvas, and I. Nitu. “TERMOS: a Formal Language for Scenarios in Mobile Computing Systems”. In: *Mobile and Ubiquitous Systems: Computing, Networking, and Services (MobiQuitous 2010)*. 2010, pp. 285–296. DOI: 10.1007/978-3-642-29154-8_24.