Exemplary and Complete Object Interaction Descriptions*

Ruth Breu, Radu Grosu, Christoph Hofmann,
Franz Huber, Ingolf Krüger, Bernhard Rumpe,
Monika Schmidt, Wolfgang Schwerin

email: {breur,grosu,hofmannc,huberf,kruegeri,rumpe,schmidtm,schwerin}
@informatik.tu-muenchen.de

Technische Universität München
Arcisstr. 21
D-80290 München, Germany

Due to their graphical and intuitive syntax Message Sequence Charts (MSC) [IT94, IT96] are an excellent description technique to capture interaction sequences among components. MSCs are used in a variety of modeling techniques and development phases. Examples of their wide-spread applicability are communication patterns in software architectures, as well as message flow protocols between business corporations.

In this paper, we present a variant of message sequence charts called Extended Event Traces (EETs). We provide the graphical and the textual notation, discuss the methodological use of EETs to describe behavior of object-oriented business information systems, and sketch their semantics. Special emphasis is put on the different implications of using EETs for exemplary and complete interaction descriptions.

The possibility to describe interactions between single objects as well as composite objects with EETs makes them particularly suitable to describe the behavior of large systems.

1 Introduction and Classification

Message sequence diagrams are one of the most widely accepted graphical techniques for describing the dynamic behavior of systems. Originally developed for the design

* This paper is joint work of the members of the projects SysLab (supported by the DFG under the Leibniz Program, and by Siemens-Nixdorf), Arcus (supported by the BMBF under project name 'ENTSTAND'), and Bellevue (supported by the DFG).

[BGH+98] R. Breu, R. Grosu, C. Hofmann, F. Huber, I. Krüger, B. Rumpe, M. Schmidt, W. Schwerin.
Exemplary and Complete Object Interaction Descriptions.
In: Computer Standards & Interfaces. Volume 19, p. 335-345. 1998.
www.se-rwth.de/publications

of technical systems (in particular: telecommunication protocols, cf. [IT94, IT96]), sequence diagrams have recently become increasingly popular in the field of business information systems. Most object-oriented analysis and design techniques offer some notational variant of sequence diagrams to express object interaction [RBP+91, Boo94, J+92, BRJ96].

Although most developers consider sequence diagrams intuitive and easy to draw, the exact interpretation and systematic use of these diagrams is often neglected. In this paper we both discuss the methodological use of sequence diagrams within the design process for information systems and give an overview of how sequence diagrams can be assigned an exact, formal semantics.

The adequate semantic interpretation of sequence diagrams depends on various factors. One fundamental issue is the underlying mechanism for message exchange (synchronous vs. asynchronous). While the ability to model asynchronous message transmission is of the essence for telecommunication protocols, synchronous communication might be a more adequate model for certain classes of business information systems.

Another fundamental issue (especially in the context of object-oriented modeling) addresses the question of how sequence diagrams determine the life cycles of objects. In order to answer this question, the role of sequence diagrams within the design process has to be understood. We identify two substantially different ways of how sequence diagrams can be used.

One possibility is to use sequence diagrams for describing exemplary behavior of objects (*scenarios*). Scenario-based design of systems has evolved as an excellent paradigm both to explore the requirements of a system and to communicate with customers in an appropriate way. Here, a sequence diagram describes a scenario as a single trace of events, i.e. a trace of messages exchanged by the objects of the system under design.

Alternatively, sequence diagrams can also be used to describe complete object behavior, i.e. the set of all possible interaction sequences during the lifetime of the modeled system components. It is clear that the basic form of sequence diagrams (providing only simple syntactical elements for message exchange) is not powerful enough to describe complete object behavior, and a set of additional operators has to be defined. Repetition, alternative, and hierarchical structuring operators are of particular importance. Among more elaborate extensions are recursion or feed-back constructs and conditions on object states. Notations that provide some or all of these operators are MSC96 [IT96], EETs [BHKS97, SHB96], and also most of the sequence diagrams supported by the OOA methods mentioned above.

Our claim is that these two views of sequence diagrams require a different semantic interpretation and address different design issues.

Scenarios describe single sequences of messages. Their semantics has to cope with the question of how exemplary object behavior can be interpreted in the context of complete object behavior. This question is of importance when combining different description techniques such as sequence diagrams and state transition diagrams. Many object-oriented analysis methods offer both of these description techniques but do not provide adequate guidelines, let alone a semantic foundation, for their consistency and combination.

In contrast, sequence diagrams used for a complete description of object behavior impose conditions on the overall set of messages that an object can send or receive. Since the notations mentioned earlier support complex constructs, such as repetition and alternative, the semantic foundation of these constructs helps designers to obtain a better understanding of the diagrams they develop. Other important issues are the combination of different diagrams involving the same set of objects and the refinement of diagrams during design.

In summary, the aim of this paper is to discuss the semantical and methodological questions that arise when sequence diagrams are applied in the design process for object-oriented business information systems.

For the semantic model we use our experience gained while working on the semantic foundation of software engineering techniques, and software architecture using formal methods. In particular, our group has developed a formal model, called *system model* (cf. [KRB96]), which is capable of modeling various system views that arise during the development process. These views are modeled abstractly and independently of specific description techniques.

The structure of this paper is as follows. In Section 2 we provide examples of the application of sequence diagrams as a description technique for exemplary (Section 2.1) and complete object interactions (Section 2.2); furthermore, we give a precise textual representation of the graphical syntax. In Section 3 we give a formal semantics for EETs. We discuss our conclusions in Section 4.

2 Notation and Methodology

The basic constituents of sequence diagrams are components and events. In general, a component may be a single object or a composite object performing a certain task. Henceforth, we use the notion of component and object interchangeably. For analysis of business requirements, people, departments, or companies may be regarded as components.

Each object that participates in an interaction is depicted by a vertical axis (labeled with the object name) that represents (part of) the lifetime of the object where time advances from top to bottom. An interaction is indicated by an arrow directed from the sending object to the receiver. Interactions are labeled with a message name and an optional list of parameters.

Figure 1 depicts a first view of a car rental company, an example that we will employ throughout this paper. The given sequence diagram models a customer making a car reservation at some reservation branch.

The diagram of Figure 1 can be interpreted in a number of different ways:

- Customer and Reservation Branch are able to exchange and to react to the given sequence of messages.
- Every time Customer sends message *request* to Reservation Branch, the messages *offer* and *confirmation* have to be exchanged consecutively (with the variation that

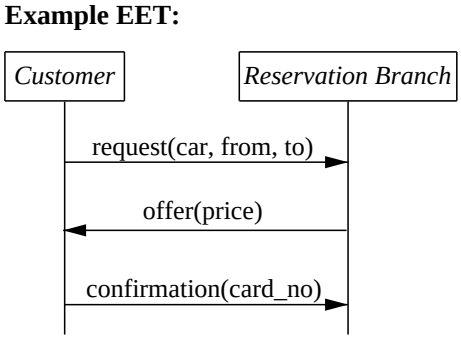

Figure 1: Simplified View of a Car Reservation

Customer and Reservation Branch may or may not send or receive other messages in between).

- The diagram describes the full interaction between Customer and Reservation Branch, i.e. both objects can only interact observing the protocol defined by the sequence of messages given in the diagram.

The above interpretations differ in the degree in which they constrain the overall behavior of the objects involved. Most analysis methods leave open how sequence diagrams are interpreted in their framework. Since an analysis model is a kind of contract either between the customer and the developer or the designer and the implementor, severe misunderstandings may be the consequence.

On the other hand, our observation is that there is no "best" interpretation of sequence diagrams and their semantics is heavily influenced by the context in which they are used. For this reason, the rest of this section deals with the methodological aspects of design with sequence diagrams. Section 2.1 focuses on sequence diagrams describing exemplary behavior of objects while Section 2.2 is concerned with the description of complete interaction behavior. To discuss the semantics of sequence diagrams more precisely, we provide a textual syntax in Section 2.3.

2.1 Exemplary Descriptions

In Figure 2 an interaction pattern of a successful car reservation is depicted. Interpreting the given sequence diagram as an exemplary scenario, i.e. as one particular sequence of interactions possibly occuring in the system under specification, the following system properties are expressed:

- Structurally, the system encompasses at least one reservation branch component, one pickup branch component, and one customer component.
- With respect to the system's behavior the following must be true: Upon receiving a *request* message with three formal parameters (of some types that are not

specified here), *one possible* reaction of Reservation Branch is the emission of a *check_availability* message to Pickup Branch. If the receiving pickup branch component replies with message *available* then the reservation branch can send an *offer* message to the requesting customer, and so on.

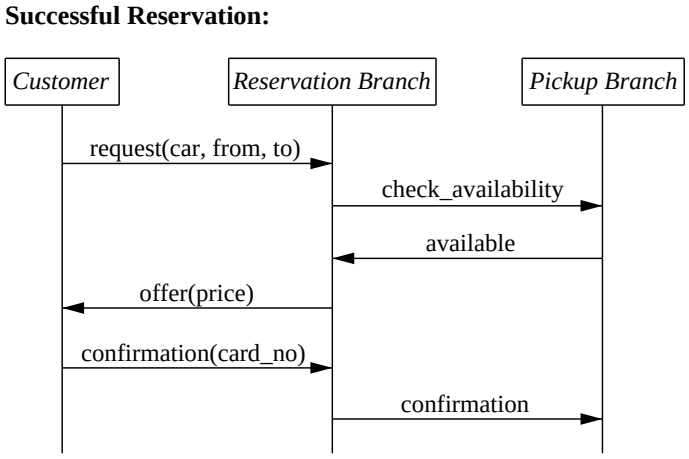

Figure 2: Exemplary Description of a Car Reservation

Assigning a meaning to sequence diagrams in the way described here corresponds to a loose semantic interpretation (cf. Section 3.3).

From a methodological point of view, scenarios may serve as a basis for complete interaction specifications; the latter may, for instance, be obtained by composing the former (cf. Section 2.2). Scenarios may also be used to derive more detailed behavior specifications, such as state transition diagrams.

2.2 Transition to Complete Behavioral Descriptions

The previous section outlines one typical scenario when a customer reserves a car in a reservation branch. The diagram in Figure 2 depicts the interactions that occur when the requested car is available, and the customer accepts the price. In order to give a more complete description of the possible interactions in a reservation branch, i.e. all possible interactions between a customer, the reservation branch, and the pickup branch, we first have to analyze whether there are other possible interaction sequences.

In the current example, we identify two additional scenarios, depicted in Figure 3 (a) and (b). Figure 3 (a) describes the scenario when a customer wishes to reserve a car that is not available for the requested period. Figure 3 (b) treats the situation where the car is available, but the customer does not accept the price.

Now, having specified all relevant scenarios (of this simplified example), we have to clarify the relationships between them, i.e. we have to analyze in which sequence they may occur. Afterwards, the scenarios must be composed, according to the sequences

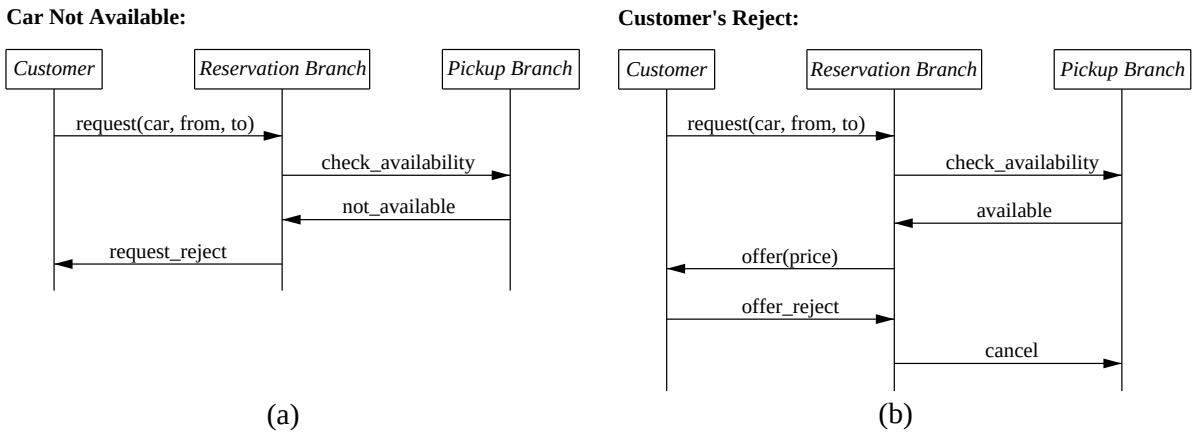

Figure 3: Exemplary Descriptions of an Unsuccessful Car Reservation

found, i.e. the particular scenarios have to be combined to more general ones. For that purpose, we use the following operators that are provided by EETs:

Sequential composition allows concatenation of a sequence of scenarios.

Finite Choice (represented by a box referencing a finite set of scenario descriptions) specifies that, when the box is to be unfolded, one element from the set has to be chosen.

Repetition (indicated by a line labeled with the number of iterations to the right of the EET) denotes part of a scenario description that may occur optionally or repeatedly. The vertical dimension of the indicator designates the operator's scope.

Based on the assumption that, in our example, a customer can either make (at most) one successful reservation after an arbitrary number of unsuccessful attempts we obtain the diagram depicted in Figure 4. This diagram does not describe a single scenario, but the set of all possible interaction sequences. First of all, there may be a finite number of instantiations of the box labeled with *Failed Reservation*. Each time the box is instantiated one scenario contained in the set is chosen. After that, there may be zero or one instantiations of the box labeled *Successful Reservation*.

2.3 Precise Syntax

To provide a precise semantics for a graphical modeling technique, first a precise definition of the syntax must be provided. For UML, this was e.g. done in document [BRJ96] (irritatingly called "UML semantics"). For textual notations, usually a concrete or an abstract grammar is given. For graphical notations, like our EETs, we could use graph grammars, a mathematical description based on tuples, relations and maps. Instead, we provide a precise definition of a textual version of EETs by using a grammar. The

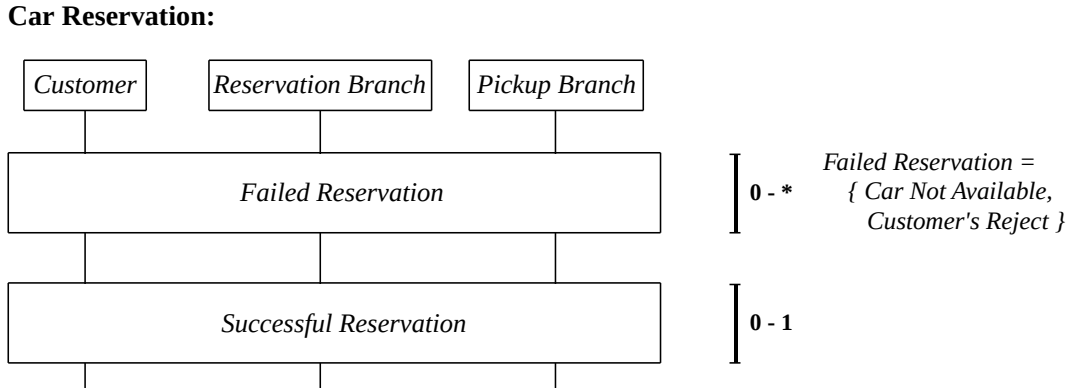

Figure 4: Complete Description of a Car Reservation

translation of EETs is done in such a way, that the graphical and the textual representations are isomorphic. Therefore, we henceforth identify EETs with their corresponding textual representation.

EETs have the following (abstract) BNF grammar (terminal symbols are highlighted by underscores):

$$
\begin{array}{lcl}
\langle \text{EETS} \rangle & ::= & \underline{\textbf{empty}} \\
& | & \langle \text{Msg} \rangle \\
& | & \langle \text{EETS} \rangle \ [\ \underline{;} \mid \underline{|} \mid \underline{\simeq}_{\langle \text{Msg} \rangle^*}\ ]\ \langle \text{EETS} \rangle \\
& | & \langle \text{EETS} \rangle \underline{^*}
\end{array}
$$

The syntactic category ⟨Msg⟩ is used to represent the messages that constitute the interactions between components.

$$
\begin{array}{lcl}
\langle \text{Msg} \rangle & ::= & \underline{(}\ \langle \text{ComponentName} \rangle \underline{,}\ \langle \text{ComponentName} \rangle \underline{,}\ \langle \text{MsgH} \rangle\ \underline{)} \\
\langle \text{MsgH} \rangle & ::= & \langle \text{MessageName} \rangle \\
& | & \langle \text{MessageName} \rangle \underline{,}\ \underline{(}\ \langle \text{FPars} \rangle\ \underline{)} \\
\langle \text{FPars} \rangle & ::= & \langle \text{FParName} \rangle \\
& | & \langle \text{FParName} \rangle \underline{,}\ \langle \text{FPars} \rangle
\end{array}
$$

For brevity, the productions for ⟨ComponentName⟩, ⟨MessageName⟩ and ⟨FParName⟩ are omitted here. They can be represented by strings denoting component, message and formal message parameter names, respectively. The component names in a message reference the sender and the receiver of the message in that order. Thus, a message determines its sender and receiver, the message name, and an optional parameter tuple.

The definition given above is regarded to be abstract in the sense that, for a concrete textual representation, parenthesis have to be introduced to group ⟨EETS⟩ expressions, and a precedence rule should be given. Here, we assume all operators associate to the left. The operators $\underline{;}$, $\underline{|}$, and $\underline{\simeq}$ bind with equal strength. $\underline{*}$ has higher precedence than all others.

The intuitive interpretation for the operators defined in the productions of ⟨EETS⟩ will be sequential composition ($\underline{;}$), alternative ($\underline{|}$), and repetition ($\underline{*}$), as described in the previous sections.

An operator that has not been applied in the examples in Section 2 is parallel composition. Here, we do not provide a graphical representation; textually the parallel composition of ⟨EETS⟩ expressions α and β is denoted by $\alpha \sim_S \beta$. In its simplest form, where $S = \emptyset$ this operator denotes the interleaving of all message sequences obtained from α and β. However, if two ⟨EETS⟩ expressions represent overlapping interaction observations and thus share messages, a more elaborate composition mechanism is appropriate. To that end, we parameterize the parallel composition operator by a set S, containing the shared messages. For its formal definition see Section 3.2, below.

Next, we define a formal semantics for the language introduced above.

3 Semantics

In this section we discuss some of the issues that have to be addressed when assigning a formal semantics to sequence diagrams. To that end, in Section 3.1, we briefly mention the basic concepts (such as the communication primitives) of the mathematical model we target at. Section 3.2 contains a rough sketch of how EETs can be assigned a trace semantics. In Section 3.3 we explain why this semantics fits well into an integrated mathematical modeling technique that covers the entire system development process.

3.1 Basic Concepts

Most concrete (physical) systems have a particular architecture (reflected, for instance, by the system's decomposition into components and their relationships), a particular state (from the set of possible system states) and a particular behavior. As a consequence, one approach to a complete specification of such systems consists of providing several models, each being concerned with one of the above aspects. Sequence diagrams model the behavior of a system by describing the interactions among the system's components and between its components and the environment.

For the purpose of defining a semantics for a given EET we consider an *interaction* to be a message exchanged between two participating components. An EET then describes a set of *interaction sequences*. Other system properties, such as component states, are abstracted away. The only purpose of the components on this level of abstraction is to send and to receive messages. The meaning of an interaction description is an *assertion* about the possible message exchanges among the participating components. As discussed in Section 2, this assertion may describe the complete set of interaction sequences or only some constraint on the set of possible sequences.

Alternatively, one can consider *events*, i.e., particular occurrences of interactions instead of interactions and *partial orders* between events instead of interaction sequences. Moreover, one can distinguish between the event of *sending* a message and the event of

receiving a message. This approach is used for MSCs [IT96] and allows the designer to model message overtaking.

There is a close connection between the choice of atomic actions (events or interactions) and the communication paradigm of the system modeled by sequence diagrams. We distinguish between *synchronous* and *asynchronous* communication. The former allows the receiver to block the sender, i.e., the communication takes place only when both communication partners are ready to communicate. In this case sending and receiving together constitute an atomic event. In the second case, the receiver cannot block the sender, i.e., the sender can pass a message that is processed later by the receiver. This is usually accomplished by providing buffers between senders and receivers. In this case sending and receiving are different events.

The EETs as introduced above use synchronous communication while MSCs use asynchronous communication. Both paradigms have their merits and each of them is more natural for a particular problem setting. For example, when modeling a network protocol, it is more natural to consider asynchronous communication between the network nodes and synchronous communication inside each node. We have adopted the synchronous paradigm for EETs because we consider it to be particularly useful for modeling large parts of object-oriented business information systems (such as transaction management within database applications).

A special mechanism that is very important in an object-oriented setting is remote procedure call. Here, the messages are partitioned into operation invocations, which initiate an operation at the receiving object, and return messages, which represent the result of an operation. Note that the remote procedure call shares properties with both of the communication paradigms mentioned above: on the one hand the receiver may not block the sender, on the other hand the sender is blocked until the receiver is willing to deliver the result. Such basic mechanisms can be dealt with in two different ways. One approach is to encode them directly into the definition of the semantics for EETs. Another one is to represent these mechanisms as constraints on a more general semantic model (cf. Section 3.3) covering other system views as well.

3.2 Synchronous Trace Semantics

As we have already pointed out, the semantics of an interaction description strongly depends on the underlying communication paradigm. EETs are based on synchronous message exchange. In this section, we briefly describe their semantics. A more thorough semantical treatment can be found in [BHKS97, SHB96]. For the semantics of MSCs we refer the interested reader to [IT95], where it is defined in a process-algebraic setting.

The graphical constructs we consider are: empty EET, single message exchange, choice, sequential and parallel composition, and iteration. The textual syntax of these operators was introduced in the previous section. Each textual (or graphical) ⟨EETS⟩ term T defines a set of interaction sequences $P[\![T]\!]$. In the following, we identify this set with its characteristic predicate, which is defined over finite traces t, with $t \in AMsg^*$. $AMsg$ denotes the set of all possible messages where the formal parameters are replaced by concrete values. The predicate is defined inductively over the structure of T.

The semantics is given by sets of finite sequences over the message set $AMsg$. The empty sequence is denoted by ϵ, a singleton sequence obtained from message m by $\langle m \rangle$ and concatenation by $t_1 \circ t_2$. Where appropriate, we omit brackets ($\langle . \rangle$ and $\{.\}$) for reasons of readability.

Empty EET. An empty EET describes a single interaction sequence: the empty sequence. It is used to express the absence of communication among a set of components.

$$P[\![\underline{\textbf{empty}}]\!].t \;\equiv\; t = \epsilon$$

Message EET. An EET denoting a single message exchange defines a set of singleton interaction sequences, each interaction containing a message where every formal parameter is instantiated with a value of the parameter's domain.

$$P[\![m]\!].t \;\equiv\; \exists m' \in \mathrm{dom}.m : t = \langle m' \rangle$$

Choice. The operator representing the choice between two EETs has the union of the sets of interaction sequences described by each of the EETs as its semantics.

$$P[\![\alpha \underline{|} \beta]\!].t \;\equiv\; P[\![\alpha]\!].t \vee P[\![\beta]\!].t$$

Sequential composition. The sequential composition of two EETs describes the set of interaction sequences obtained by collecting all possible concatenations of interaction sequences denoted by the first EET with interaction sequences denoted by the second one.

$$P[\![\alpha \underline{;} \beta]\!].t \;\equiv\; \exists t_0, t_1 \in AMsg^* : P[\![\alpha]\!].t_0 \wedge P[\![\beta]\!].t_1 \wedge t = t_0 \circ t_1$$

Iteration. The iteration operator applied to an EET yields the set of interaction sequences obtained by concatenating an arbitrary number of times the interaction sequences of the operand EET.

$$P[\![\alpha^{\underline{*}}]\!].t \;\equiv\; \exists n \geq 0 \; \exists\, t_1, \ldots, t_n \in AMsg^* :\; t = t_1 \circ \ldots \circ t_n \;\wedge\; P[\![\alpha]\!].t_1 \;\wedge\; \ldots \;\wedge\; P[\![\alpha]\!].t_n$$

Parallel composition. The parallel composition of two EETs describes the set of interaction sequences obtained by collecting all possible interleavings of the interaction sequences of the operand EETs. If both EETs share messages having the same sender and receiver (like EET α and β in Figure 5) then two possible interpretations of the parallel composition operator seem senseful:

- Any resulting EET contains all messages of both operand EETs, i.e. the number of messages in any resulting EET is the sum of messages contained in the operand EETs (consider EET ρ in Figure 5, which is one element of the resulting set of interaction sequences).

- Any resulting EET contains all messages of both operand EETs, where all shared messages are included only once (consider EET γ in Figure 5, where message a appears only once). Here, shared messages are identified in the resulting EET.

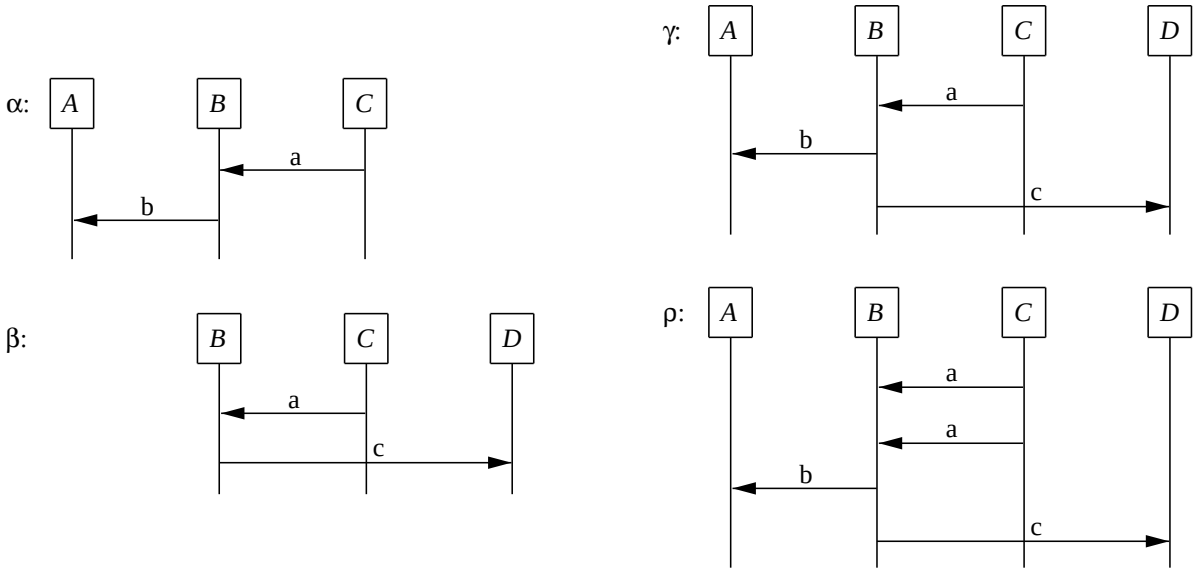

Figure 5: Parallel Composition of EETs: Two Possible Interpretations

To support the use of both interpretations, we define a parameterized parallel composition operator $\sim_S$ that allows the designer to explicitly describe the set of messages S that are identified in both EETs. To define the semantics, we first introduce an oracle-construct, which allows us to select from a given stream t, where $t \in AMsg^*$, a substream of messages t^{ϕ}_{M}, by using an oracle $\phi \in \{1, 2, Both\}^*$ and a filter set $M \subseteq \{1, 2, Both\}$. For each message in t the oracle ϕ tells us where the message belongs to:

$$\epsilon^{\phi}_{M} = \epsilon$$
$$(t_1 \circ t)^{\phi_1 \circ \phi}_{M} = \begin{cases} t_1 \circ t^{\phi}_{M} & \text{if} \quad \phi_1 \in M \\ t^{\phi}_{M} & \text{if} \quad \phi_1 \notin M \end{cases}$$

We define the following semantics for the parallel composition operator:

$$\begin{aligned} P[\![\alpha \underline{\sim}_S \beta]\!].t \;\equiv\; & \exists \phi \in \{1, 2, Both\}^*,\, t_1, t_2 \in AMsg^* : \\ & P[\![\alpha]\!].t_1 \;\wedge\; P[\![\beta]\!].t_2 \wedge \\ & t_1 = t^{\phi}_{\{1,Both\}} \wedge t_2 = t^{\phi}_{\{2,Both\}} \wedge \\ & t^{\phi}_{\{Both\}} \in S^* \wedge t^{\phi}_{\{1,2\}} \in (AMsg \setminus S)^* \end{aligned}$$

Please note that the messages from set S have to occur in the same order and number in both operand EETs. Otherwise the EETs are incompatible. Also note that sharing of messages can only occur if at least two components are shared in both EETs, α and

β. If it is intended not to identify any message from the operand EETs, $\sim_\emptyset$ is used. For that purpose, we introduce the shortcut $\sim$. The semantics then simplifies to:

$$\begin{aligned} P[\![\alpha \sim \beta]\!].t \;\equiv\; & \exists \phi \in \{1,2\}^*,\, t_1, t_2 \in AMsg^* : \\ & P[\![\alpha]\!].t_1 \;\wedge\; P[\![\beta]\!].t_2 \wedge \\ & t_1 = t^{\phi}_{\{1\}} \;\wedge t_2 = t^{\phi}_{\{2\}} \end{aligned}$$

To increase the expressive power of the description technique we may use predicates to constrain the sets of interaction sequences allowed by an EET.

One can also consider extensions of EETs by taking component states into account and by using these states to trigger interactions. Moreover, one can imagine additional composition mechanisms and several notions of refinement for EETs. We currently also examine an adequate method-call operator suitable for modeling object-oriented systems with EETs, as well as other "special purpose" operators.

3.3 Integration of the Semantics

One of the most important issues that has to be addressed when introducing a certain description technique is to define its relationship with existing techniques. This is especially true if the latter's scope is not orthogonal to the former's. Then, the definition and maintenance of consistency among the results of applying different notations is a prerequisite for assuring correctness in the development process.

One approach that facilitates providing an integrated semantics for different description techniques has been explored in the SysLab project. Here, syntactic diagrams (also called documents in this context) are assigned a semantics by defining a mapping that relates each diagram to a set of systems. The notion of a system is defined using an abstract mathematical model (cf. [KRB96], an extension appears in [GKR96]). The description of a system consists of various documents that describe, for example, the properties of classes, objects, and object behavior. Every document represents a certain view of the system. One such view is the semantic model of object interaction as described in the previous sections.

The notion of consistency of interaction descriptions with other description techniques can be defined by providing mappings that extract the communication information from the latter. The result of this extraction has to be matched against the set of interaction sequences as defined by the EET document. At this point the distinction between exemplary and complete interaction descriptions comes into play. If we use EETs to represent scenarios, the matching process has to ensure the following: in the set of interaction sequences extracted from the other documents there must be at least one element that is equivalent to the semantics of the scenario. This corresponds to a loose semantic interpretation. Complete interaction descriptions, on the other hand, yield another notion of consistency: Every element of the set of extracted interaction scenarios must be contained in the semantics of the corresponding EET document. In either case an EET document may be seen as a set of proof obligations that have to be discharged to establish the consistency among the documents representing the system.

Because the semantics of documents, such as EETs, is given by the set of all systems that obey the induced properties each document can be seen as a proposition about the system to be implemented. Several documents can then be combined at the semantics level by conjoining their semantics (which corresponds to intersection on sets). Refinement and composition also have a semantic counterpart: they result in set inclusion. This means that adding information to a document always results in a smaller set of systems. For a complete integration of EETs into the system development process we have to define adequate notions for their refinement. This is an active area of work in our group.

4 Conclusion

In the previous sections we have both explored the application of sequence diagrams as a description technique for exemplary and complete object interactions, and outlined aspects of their formal foundation. Each of the two different semantic interpretations has its specific methodological implications.

The use of sequence diagrams as a complete description of object interactions leads to the need of operators combining sequence diagrams in various ways. To that purpose, we have defined operators for choice, sequential composition, iteration and parallel composition of sequence diagrams. In particular the latter is a powerful tool to combine different but overlapping views on object interactions.

There is ample opportunity for future research in the following areas. First, we are investigating steps towards providing developers with a methodological framework for the transition from exemplary to complete interaction descriptions. Second, we explore the combination of interaction descriptions with other description techniques, like state transition diagrams, to integrate the former into a more general design context. This requires studying adequate notions of consistency among these notations. Together with the concepts presented in this paper, these steps form the basis for integrated tools supporting developers in the design of large, object-oriented business information systems.

References

[BHKS97] Manfred Broy, Christoph Hofmann, Ingolf Krüger, and Monika Schmidt. A graphical description technique for communication in software architectures. TUM-I 9705, Technische Universität München, 1997.

[Boo94] G. Booch. *Object-Oriented Analysis and Design with Applications.* Addison Wesley, 2nd edition, 1994.

[BRJ96] G. Booch, J. Rumbaugh, and I. Jacobson. The Unified Modeling Language for Object-Oriented Development, Version 1.0, 1996.

[GKR96] Radu Grosu, Cornel Klein, and Bernhard Rumpe. Enhancing the syslab system model with state. TUM-I 9631, Technische Universität München, 1996.

[IT94] ITU-T. *Z.120 – Message Sequence Chart (MSC).* ITU-T, Geneva, 1994.

[IT95] ITU-T. *Z.120 B – Message Sequence Chart Algebraic Semantics.* ITU-T, Geneva, 1995.

[IT96] ITU-T. *Z.120 – Message Sequence Chart (MSC).* ITU-T, Geneva, 1996.

[J+92] I. Jacobson et al. *Object-Oriented Software Engineering.* ACM Press, 1992.

[KRB96] C. Klein, B. Rumpe, and M. Broy. A stream-based mathematical model for distributed information processing systems - SysLab system model - . In Jean-Bernard Stefani Elie Naijm, editor, *FMOODS'96 Formal Methods for Open Object-based Distributed Systems*, pages 323–338. ENST France Telecom, 1996.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice Hall, 1991.

[SHB96] Bernhard Schätz, Heinrich Hußmann, and Manfred Broy. Graphical Development of Consistent System Specifications . In James Woodcock Marie-Claude Gaudel, editor, *FME'96: Industrial Benefit and Advances In Formal Methods*, pages 248–267. Springer, 1996. Lecture Notes in Computer Science 1051.