

A generalization between actors is shown by a generalization arrow; that is, a solid line with a closed, hollow arrow head. The arrow head points at the more general actor.

#### 3.58.3 Example

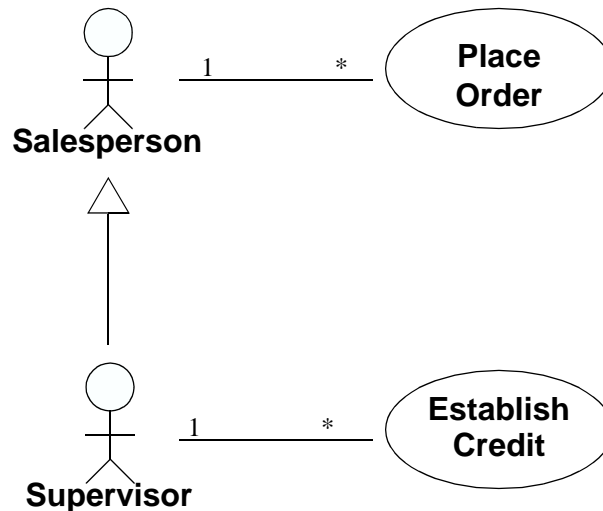


Figure 3-54 Actor Relationships

#### 3.58.4 Mapping

A generalization between two actor symbols and an association between actor symbol and a use case symbol maps into the corresponding relationship between the corresponding Elements, as described above.

## Part 7 - Interaction Diagrams

The description of behavior involves two aspects: 1) the structural description of the participants and 2) the description of the communication patterns. The structure of Instances playing roles in a behavior and their relationships is called a *Collaboration*. The communication pattern performed by Instances playing the roles to accomplish a specific purpose is called an *Interaction*. The two aspects of behavior are often described together on a single diagram, but at times it is useful to describe the structural aspects separately.

Interaction diagrams come in two forms based on the same underlying information, specified by a Collaboration and possibly by an Interaction, but each form emphasizes a particular aspect of it. The two forms are *sequence diagrams* and *collaboration diagrams*. A sequence diagram shows the explicit sequence of communications and is better for real-time specifications and for complex scenarios. A collaboration diagram shows an Interaction organized around the roles in the Interaction and their

relationships. It does not show time as a separate dimension, so the sequence of communications and the concurrent threads must be determined using sequence numbers.

## 3.59 Collaboration

### 3.59.1 Semantics

Behavior is implemented by ensembles of Instances that exchange Stimuli within an overall interaction to accomplish a task. To understand the mechanisms used in a design, it is important to see only those Instances and their cooperation involved in accomplishing a purpose or a related set of purposes, projected from the larger system of which they are part of. Such a static construct is called a *Collaboration*.

A Collaboration includes an ensemble of ClassifierRoles and AssociationRoles that define the participants needed for a given set of purposes. Instances conforming to the ClassifierRoles play the roles defined by the ClassifierRoles, while Links between the Instances conform to AssociationRoles of the Collaboration. ClassifierRoles and AssociationRoles define a usage of Instances and Links, and the Classifiers and Associations declare all required properties of these Instances and Links.

An *Interaction* is defined in the context of a Collaboration. It specifies the communication patterns between the roles in the Collaboration. More precisely, it contains a set of partially ordered *Messages*, each specifying one communication; for example, what Signal to be sent or what Operation to be invoked, as well as the roles to be played by the sender and the receiver, respectively.

A *CollaborationInstanceSet* references an ensemble of Instances that jointly perform the task specified by the CollaborationInstanceSet's Collaboration. These Instances play the roles defined by the ClassifierRoles of the Collaboration; that is, the Instances have all the properties declared by the ClassifierRoles (the Instances are said to *conform to* the ClassifierRoles). The Stimuli sent between the Instances when performing the task are participating in the *InteractionInstanceSet* of the CollaborationInstanceSet. These Stimuli conform to the Messages in one of the Interactions of the Collaboration. Since an Instance can participate in several CollaborationInstanceSets at the same time, all its communications are not necessarily referenced by only one InteractionInstanceSet. They can be interleaved.

A Collaboration may be attached to an Operation or a Classifier, like a UseCase, to describe the context in which their behavior occurs; that is, what roles Instances play to perform the behavior specified by the Operation or the UseCase. A Collaboration is used for describing the realization of an Operation or a Classifier. A Collaboration that describes a Classifier, like a UseCase, references Classifiers and Associations in general, while a Collaboration describing an Operation includes the arguments and local variables of the Operation, as well as ordinary Associations attached to the Classifier owning the Operation. The Interactions defined within the Collaboration specify the communication pattern between the Instances when they perform the behavior specified in the Operation or the UseCase. These patterns are presented in

sequence diagrams or collaboration diagrams. A Collaboration may also be attached to a Class to define the static structure of the Class; that is, how attributes, parameters, etc. cooperate with each other.

A parameterized Collaboration represents a design construct that can be used repeatedly in different designs. The participants in the Collaboration, including the Classifiers and Relationships, can be parameters of the generic Collaboration. The parameters are bound to particular ModelElements in each instantiation of the generic Collaboration. Such a parameterized Collaboration can capture the structure of a *design pattern* (note that a design pattern involves more than structural aspects). Whereas most Collaborations can be anonymous because they are attached to a named ModelElement, Collaboration patterns are free standing design constructs that must have names.

A Collaboration may be expressed at different levels of granularity. A coarse-grained Collaboration may be refined to produce another Collaboration that has a finer granularity.

### 3.60 Sequence Diagram

#### 3.60.1 Semantics

A sequence diagram presents an Interaction, which is a set of Messages between ClassifierRoles within a Collaboration, or an InteractionInstanceSet, which is a set of Stimuli between Instances within a CollaborationInstanceSet to effect a desired operation or result.

#### 3.60.2 Notation

A sequence diagram has two dimensions: the vertical dimension represents time, and the horizontal dimension represents different instances. Normally time proceeds down the page. (The dimensions may be reversed, if desired.) Usually only time sequences are important, but in real-time applications the time axis could be an actual metric. There is no significance to the horizontal ordering of the instances.

The different kinds of arrows used in sequence diagrams are described in Section 3.63, “Message and Stimulus,” on page 3-111. These are the same kinds as in collaboration diagrams; see Section 3.65, “Collaboration Diagram,” on page 3-114.

Note that much of this notation is drawn directly from the Object Message Sequence Chart notation of Buschmann, Meunier, Rohnert, Sommerlad, and Stal, which is itself derived with modifications from the Message Sequence Chart notation.

#### 3.60.3 Presentation Options

The horizontal ordering of the lifelines is arbitrary. Often call arrows are arranged to proceed in one direction across the page; however, this is not always possible and the ordering does not convey information.

The axes can be interchanged, so that time proceeds horizontally to the right and different objects are shown as horizontal lines.

Various labels (such as timing constraints, descriptions of actions during an activation, and so on) can be shown either in the margin or near the transitions or activations that they label.

Timing constraints may be expressed using time expressions on message or stimuli names. The functions *sendTime* (the time at which a stimulus is sent by an instance) and *receiveTime* (the time at which a stimulus is received by an instance) may be applied to stimuli names to yield a time. The set of time functions is open-ended, so that users can invent new ones as needed for special situations or implementation distinctions (such as *elapsedTime*, *executionStartTime*, *queuedTime*, *handledTime*, etc.)

Construction marks of the kind found in blueprints can be used to indicate a time interval to which a constraint may be attached (see bottom right of Figure 3-55 on page 3-104). This notation is visually appealing but it is ambiguous if the arrow is horizontal, because the send time and the receive time cannot be distinguished. In many cases the transmission time is negligible, so the ambiguity is harmless, but a tool must nevertheless map such a notation unambiguously to an expression on message or stimuli names (as shown in the examples in the left of the diagram) before the information is placed in the semantic model. (A tool may adopt defaults for this mapping.) Similarly, a tool might permit the time function to be elided and use the stimulus name to denote the time of stimulus sending or receipt within a timing expression (such as “b.receiveTime - a.sendTime < 1 sec.” in Figure 3-55), but again this is only a surface notation that must be mapped to a proper time expression in the semantic model).

3.60.4 Example

Simple sequence diagram with concurrent objects.

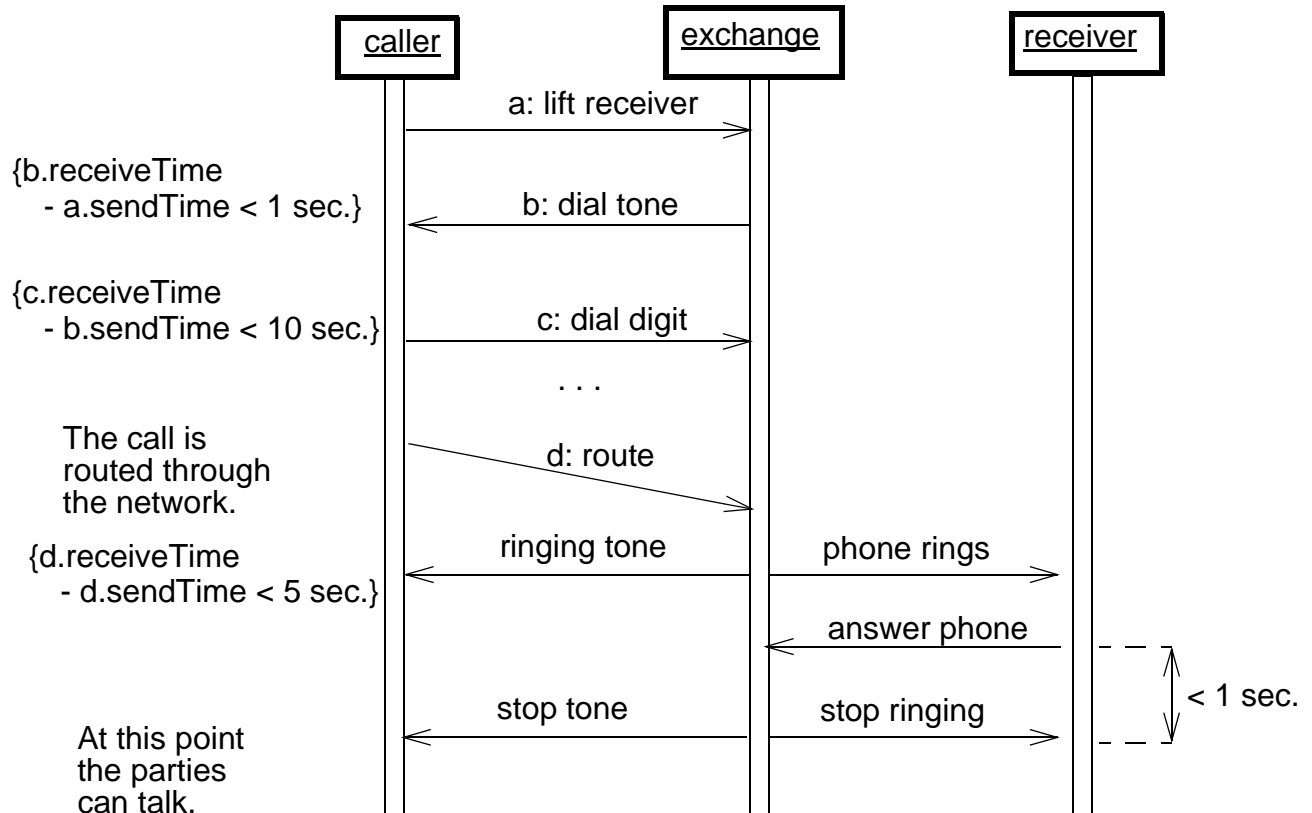


Figure 3-55 Simple Sequence Diagram with Concurrent Objects (denoted by boxes with thick borders).

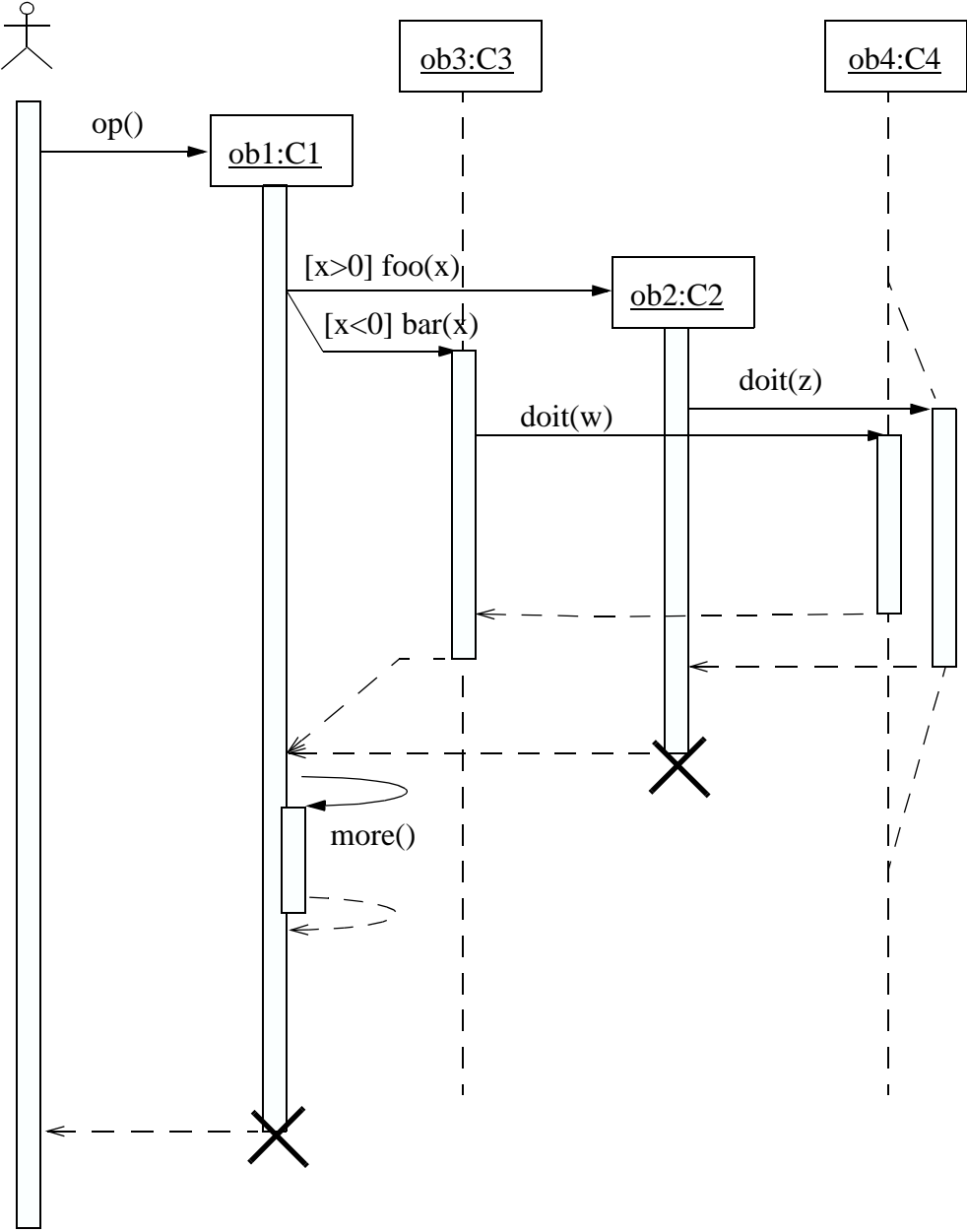


Figure 3-56 Sequence Diagram with Focus of Control, Conditional, Recursion, Creation, and Destruction.

### 3.60.5 Mapping

This section summarizes the mapping for the sequence diagram and the elements within it, some of which are described in subsequent sections.

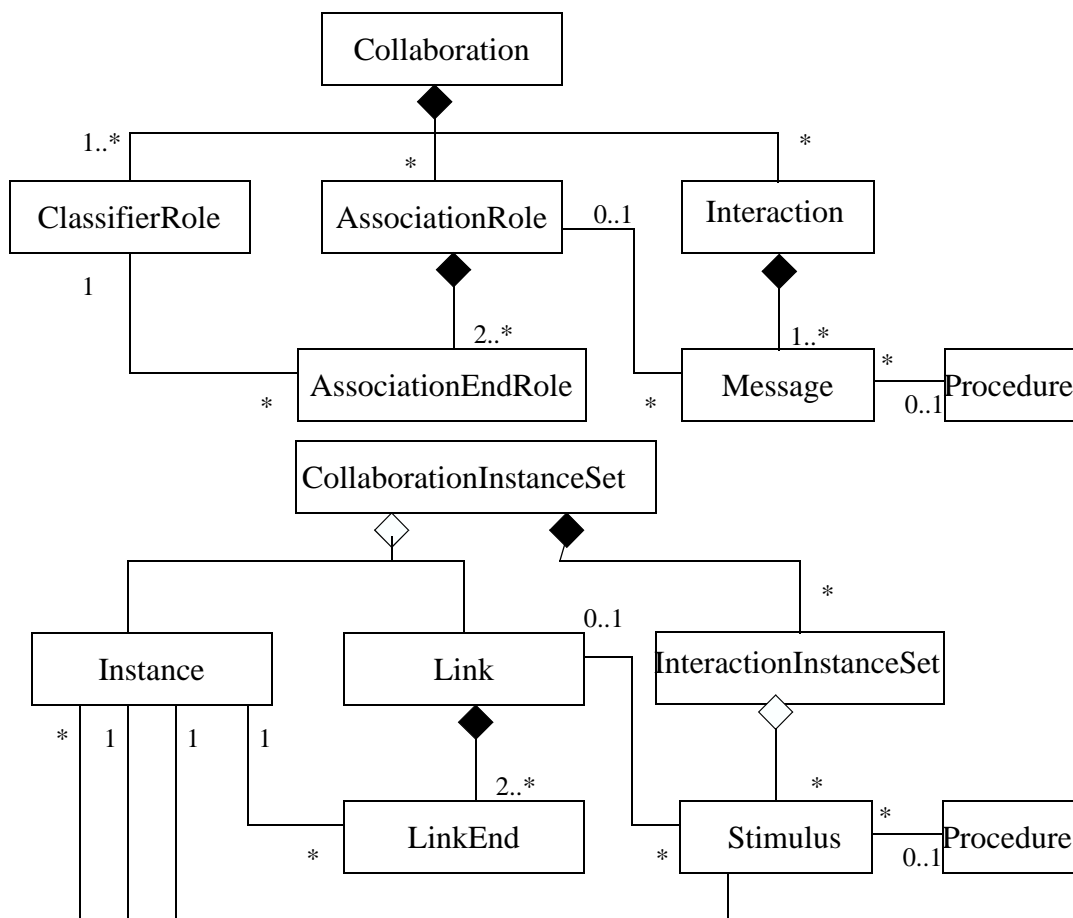


Figure 3-57 A summary of the UML constructs used in the section below.

#### 3.60.5.1 Sequence diagram

A sequence diagram maps into an Interaction and an underlying Collaboration or an InteractionInstanceSet and an underlying CollaborationInstanceSet depending on whether the diagram shows Instances or ClassifierRoles. An Interaction specifies a sequence of communications; it contains a collection of partially ordered Messages, each specifying a communication between a sender role and a receiver role. A CollaborationInstanceSet references a collection of Instances that conform to the ClassifierRoles in the Collaboration owning the Interaction. These Instances communicate by dispatching Stimuli that conform to the Messages in the Interaction. The CollaborationInstanceSet has an InteractionInstanceSet that references these Stimuli. A sequence diagram presents either a collection of object symbols and arrows

mapping to Instances and Stimuli, or a collection of classifier-role symbols and arrows mapping to ClassifierRoles and Messages. The Instances and Stimuli conform to the ClassifierRoles and Messages.

The sequence diagram presents either a Collaboration or a CollaborationInstanceSet. In the former case, the classifier box with its lifeline maps onto a ClassifierRole in the Collaboration, and the arrows map onto the Messages in one of the Collaboration's Interactions. The name strings in the boxes map onto the names of the ClassifierRoles, while the classifier names map onto the ClassifierRole's *base* Classifiers. The AssociationRoles among the ClassifierRoles are not shown on the sequence diagram. They must be obtained in the model from a complementary collaboration diagram or other means.

If the sequence diagram presents a CollaborationInstanceSet, each object box with its lifeline maps into an Instance, which conforms to a ClassifierRole in the CollaborationInstanceSet's Collaboration. The name field maps into the name of the Instance, the role name into the ClassifierRole's name, and the class field maps into the names of the Classifiers being the *base* Classifiers of the ClassifierRole. An arrow maps into a Stimulus connected to two Instances: the sender and the receiver. The Link used for the communication of the Stimulus plays the role specified by the AssociationRole connected to the Message. Unless the correct Link can be determined from a complementary collaboration diagram or other means, the Stimulus is either not attached to a Link (not a complete model), or it is attached to an arbitrary Link or to a dummy Link between the Instances conforming to the AssociationRole implied by the two ClassifierRoles due to the lack of complete information.

The label of the arrow is mapped into either the body attribute of the Procedure, or into a detailed action model. For the action model, the name of the Operation to be invoked or Signal to be sent is mapped onto the name of the Operation or Signal invoked by the actions in the Procedure connected to the Message. Different alternatives exist for showing the arguments of the Stimulus. If references to the actual Instances being passed as arguments are shown, these are mapped onto the arguments of the Stimulus. If the argument expressions are shown instead, and a detailed action model is used, then these are mapped into CodeActions in the Procedure, or additional actions that compute the values of the expressions. Finally, if the types of the arguments are shown together with the name of the Operation or the Signal, these are mapped onto the parameter types of the Operation or the Attribute types of the Signal, respectively. A timing label placed on the level of an arrow endpoint maps into the name of the corresponding Message or Stimulus. A constraint placed on the diagram maps into a Constraint on the entire Interaction.

An arrow with the arrowhead pointing to an object symbol or role symbol within the frame of the diagram maps into a Stimulus (Message) dispatched by a CreateObjectAction. The interpretation is that an Instance is created by dispatching the Stimulus. If the target of the arrow is a classifier-role symbol, the Instance will conform to the ClassifierRole. (Note, that the diagram does not necessarily show from which Classifier the Instance originates; only that the newly created Instance conform to the ClassifierRole.) After the creation of the Instance, it may immediately start interacting with other Instances. This implies that the creation method (constructor, initializer) of the Instance dispatches these Stimuli. If an object termination symbol



("X") is the target of an arrow, the arrow maps into a Stimulus that will cause the receiving Instance to be removed. If the object termination symbol appears in the diagram without an incoming arrow, it maps into a Procedure containing a DestroyObjectAction.

The order of the arrows in the diagram maps onto pairs of associations between the Stimuli (Messages). A *predecessor* relationship is established between Stimuli (Messages) corresponding to successive arrows in the vertical sequence. In case of concurrent arrows preceding an arrow, the corresponding Stimulus (Message) has a collection of predecessors. Moreover, each Stimulus (Message) has an *activator* association to the Stimulus (Message) corresponding to the incoming arrow of the activation.

### ***Procedural sequence diagram***

On a procedural sequence diagram (one with focus of control and calls), subsequent arrows on the same lifeline map into Stimuli (Messages) obeying the *predecessor* association. An arrow to the head of a focus of control region establishes a nested activation. The arrow maps into a Stimulus (Message) with the dispatching Procedure containing a CallOperationAction. The Stimulus holds the sender and receiver Instance, as well as the argument Instances, to be supplied in the invocation and references the target Operation to be invoked. The expressions that evaluate to the arguments of the Operation are, in a detailed action model, mapped into CodeActions in the Procedure connected to the Stimulus, or additional actions that compute the values of the expressions. In the case the arrow maps onto a Message the sender and the receiver are specified by the *sender* and *receiver* ClassifierRoles of the Message. The sender and receiver Instances of a Stimulus conform to these ClassifierRoles. Any condition or iteration expression attached to the arrow becomes, in a detailed action model, the test clause action in a ConditionalAction or LoopAction in the dispatching Procedure. All arrows departing the nested activation map into Stimuli (Messages) with an *activation* Association to the Stimulus (Message) corresponding to the arrow at the head of the activation. A return arrow departing the end of the activation maps into a Stimulus (Message) with:

- an *activation* Association to the Stimulus (Message) corresponding to the arrow at the head of the activation, and
- a *predecessor* association to the previous Stimulus (Message) within the same activation; that is, the last Stimulus (Message) being sent in the activation.

A return must be the final Stimulus (Message) within a predecessor chain. It is not the predecessor of any Stimulus (Message).

## 3.61 Object Lifeline

### 3.61.1 Semantics

In a sequence diagram an object lifeline denotes an Instance playing a specific role. Arrows between the lifelines denote communication between the Instances playing those roles. Within a sequence diagram the existence and duration of the Instance in a

role is shown, but the relationships among the Instances are not shown. The role is specified by a ClassifierRole; it describes the properties of an Instance playing the role and describes the relationships an Instance in that role has to other Instances.

### 3.61.2 Notation

An Instance is shown as a vertical dashed line called the “lifeline.” The lifeline represents the existence of the Instance at a particular time. If the Instance is created or destroyed during the period of time shown on the diagram, then its lifeline starts or stops at the appropriate point; otherwise, it goes from the top to the bottom of the diagram. An object symbol is drawn at the head of the lifeline. If the Instance is created during the diagram, then the arrow, which maps onto the Stimulus that creates the Instance, is drawn with its arrowhead on the object symbol. If the Instance is destroyed during the diagram, then its destruction is marked by a large “X,” either at the arrow mapping to the Stimulus that causes the destruction or (in the case of self-destruction) at the final return arrow from the destroyed Instance. An Instance that exists when the transaction starts is shown at the top of the diagram (above the first arrow), while an Instance that exists when the transaction finishes has its lifeline continue beyond the final arrow.

The lifeline may split into two or more concurrent lifelines to show conditionality. Each separate track corresponds to a conditional branch in the communication. The lifelines may merge together at some subsequent point.

### 3.61.3 Presentation Options

In some cases, it is necessary to link sequence diagrams to each other; for example, it might not be possible to put all lifelines in one diagram, or a sub-sequence is included in several diagrams; hence, it is convenient to put the common sub-sequence in a separate diagram, which is referenced from the other diagrams. In these cases, the cut between the diagrams can be expressed in one of the diagrams with a dangling arrow leaving a lifeline but not arriving at another lifeline, and in the other diagram it is expressed with a dangling arrow arriving at a lifeline from nowhere. In both cases, it is recommended to attach a note stating which diagram the sequence originates from or continues in. This is purely notational. The different diagrams show different parts of the underlying Interaction.

### 3.61.4 Example

See also Figure 3-56 on page 3-105.

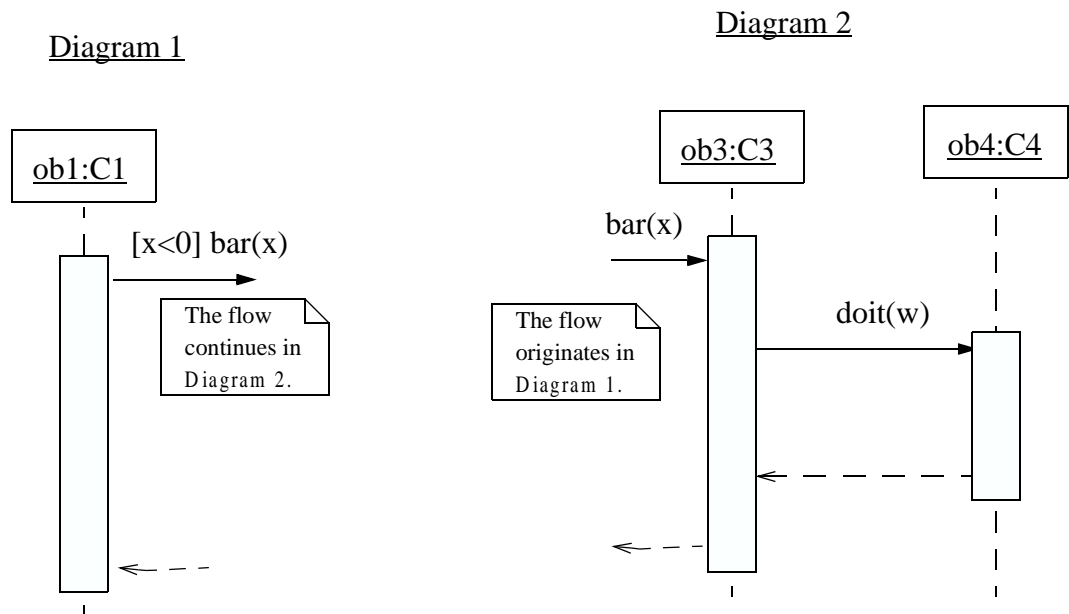


Figure 3-58 The flow shown in the sequence diagram to the left continues in the diagram to the right.

#### 3.61.5 Mapping

See Section 3.60.5, "Mapping," on page 3-106.

### 3.62 Activation

#### 3.62.1 Semantics

An activation (focus of control) shows the period during which an Instance is performing a Procedure either directly or through a subordinate procedure. It represents both the duration of the performance of the Procedure in time and the control relationship between the activation and its callers (stack frame).

#### 3.62.2 Notation

An activation is shown as a tall thin rectangle whose top is aligned with its initiation time and whose bottom is aligned with its completion time. The Procedure being performed may be labeled in text next to the activation symbol or in the left margin, depending on style. Alternately, the incoming arrow may indicate the Procedure, in which case it may be omitted on the activation itself. In procedural flow of control, the top of the activation symbol is at the tip of an incoming arrow (the one that initiates the procedure) and the base of the symbol is at the tail of a return arrow.

In the case of concurrent Instances each with their own threads of control, an activation shows the duration when each Instance is performing an Operation or transition in a state machine. Operations by other Instances are not relevant. If the distinction between direct computation and indirect computation (by a nested operation call) is unimportant, the entire lifeline may be shown as an activation.

### 3.62.3 Example

See Figure 3-55 on page 3-104 and Figure 3-56 on page 3-105.

### 3.62.4 Mapping

See Section 3.60.5, “Mapping,” on page 3-106.

## 3.63 Message and Stimulus

### 3.63.1 Semantics

A Stimulus is a communication between two Instances that conveys information with the expectation that action will ensue. A Stimulus will cause an Operation to be invoked, raise a Signal, or cause an Instance to be created or destroyed.

A Message is a specification of Stimulus, i.e. it specifies the roles that the sender and the receiver Instances must conform to, as well as the Procedure which will, when executed, dispatch a Stimulus that conforms to the Message.


### 3.63.2 Notation

In a sequence diagram a Stimulus as well as a Message is shown as a horizontal solid arrow from the lifeline of one Instance or ClassifierRole to the lifeline of another Instance or ClassifierRole. In case of a Stimulus from an Instance to itself, the arrow may start and finish on the same lifeline. The arrow is labeled with the name of the Operation to be invoked or the name of the Signal. Its argument values or argument expressions may be presented, as well.


The arrow may also be labeled with a sequence number to show the sequence of the Stimulus (Message) in the overall interaction. However, sequence numbers are often omitted in sequence diagrams, as the physical location of the arrow shows the relative sequences, but they are necessary in collaboration diagrams. Sequence numbers are useful on both kinds of diagrams for identifying concurrent threads of control. An arrow may also be labeled with a condition and/or iteration expression.

### 3.63.3 Presentation options


The following arrowhead variations may be used to show different kinds of communications.

*filled solid arrowhead* 

Operation call or other nested flow of control. The entire nested sequence is completed before the outer level sequence resumes. The arrowhead may be used to denote ordinary operation calls, but it may also be used to denote concurrently active instances when one of them sends a Signal and waits for a nested sequence of behavior to complete before it continues.

*stick arrowhead* 

Asynchronous communication; that is, no nesting of control. The sender dispatches the Stimulus and immediately continues with the next step in the execution.<sup>1</sup>

*dashed arrow with stick arrowhead* - 

Return from operation call.

***Variation:***

In a procedural flow of control, the return arrow may be omitted (it is implicit at the end of an activation). It is assumed that every call has a paired return after any subordinate stimuli. The return value can be shown on the initial arrow. For nonprocedural flow of control (including parallel processing and asynchronous messages) returns should be shown explicitly.

***Variation:***

Normally message arrows are drawn horizontally. This indicates the duration required to send the stimulus is “atomic;” that is, it is brief compared to the granularity of the interaction and that nothing else can “happen” during the transmission of the stimulus. This is the correct assumption within many computers. If the stimulus requires some time to arrive, during which something else can occur (such as a stimulus in the opposite direction), then the arrow may be slanted downward so that the arrowhead is below the arrow tail.

***Variation: Branching***

A branch is shown by multiple arrows leaving a single point, each possibly labeled by a condition. Depending on whether the conditions are mutually exclusive, the construct may represent conditionality or concurrency.

---

1. UML 1.3 and previous versions included a half-stick arrowhead notation in addition to the stick arrowhead notation. This notation has been removed because the semantic distinction between the two was subtle and confusing.

**Variation: Iteration**

A connected set of arrows may be enclosed and marked as an iteration. For a generic sequence diagram, the iteration indicates that the dispatch of a set of stimuli can occur multiple times. For a procedure, the continuation condition for the iteration may be specified at the bottom of the iteration. If there is concurrency, then some arrows in the diagram may be part of the iteration and others may be single execution. It is desirable to arrange a diagram so that the arrows in the iteration can be enclosed together easily.

**Variation:**

A lifeline may subsume an entire set of objects on a diagram representing a high-level view.

**Variation:**

A distinction may be made between a period during which an Instance has a live activation and a period in which the activation is actually computing. The former (during which it has control information on a stack but during which control resides in something that it called) is shown with the ordinary double line. The latter (during which it is the top item on the stack) may be distinguished by shading the region.

### 3.63.4 Example

See Figure 3-56 on page 3-105.

### 3.63.5 Mapping

See Section 3.60.5, “Mapping,” on page 3-106.

## 3.64 Transition Times

### 3.64.1 Semantics

A Message may specify several different times; for example, a sending time and a receiving time. These are formal names that may be used within Constraint expressions. The set of different kinds of times is open-ended so that users can invent new ones as needed for special situations, such as *elapsedTime* and *startExecutionTime*. These expressions may be used in Constraints to designate specific time constraints valid for the Message.

### 3.64.2 Notation

A transition instance (such as a Stimulus or Message in a sequence diagram, a collaboration diagram, or a Transition in a state machine) may be given a name. A timing constraint is formed as an expression based on the name of the transition. For

example, if the name of a Stimulus is *stim*, its send-time is expressed by *stim.sendTime* (), and its receive-time by *stim.receiveTime* (). The timing constraint may be shown in the left margin aligned with the arrow (on a sequence diagram) or near the tail of the arrow (on a collaboration diagram). Constraints may be specified by placing Boolean expressions, possibly including time expressions, in braces on the sequence diagram.

### 3.64.3 Presentation Options

When it is clear from the context, the name of a Message or the name of a Stimulus may itself be used to denote the time at which the transition started. In cases where the performance of the transition is not instantaneous, the time at which the transition is ended may be indicated by the same name with a prime sign appended to the name.

### 3.64.4 Example

See Figure 3-55 on page 3-104.

### 3.64.5 Mapping

See Section 3.60.5, “Mapping,” on page 3-106.

## Part 8 - Collaboration Diagrams

### 3.65 Collaboration Diagram

#### 3.65.1 Semantics

A collaboration diagram presents either a Collaboration, which contains a set of roles to be played by Instances, as well as their required relationships given in a particular context, or it presents a CollaborationInstanceSet with a collection of Instances and their relationships. The diagram may also present an Interaction (InteractionInstanceSet), which defines a set of Messages (Stimuli) specifying the interaction between the Instances playing the roles within a Collaboration to achieve the desired result.

A Collaboration is used for describing the realization of an Operation or a Classifier. A Collaboration that describes a Classifier, like a UseCase, references Classifiers and Associations in general, while a Collaboration describing an Operation includes the arguments and local variables of the Operation, as well as ordinary Associations attached to the Classifier owning the Operation.

#### 3.65.2 Notation

A collaboration diagram shows a graph of either Instances linked to each other, or ClassifierRoles and AssociationRoles; it may also include the communication stated by an Interaction or InteractionInstanceSet.