

CHAPTER 10

Collaboration Diagrams

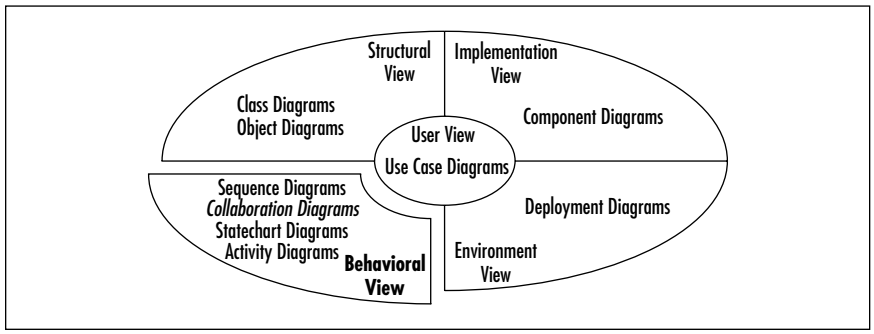


Figure 10-1: Behavioral Model View—Collaboration Diagrams

Collaboration diagrams (Figure 10-1) lie within the behavioral view of a system and render the realization and implementation of behavior. This is a definition of how behavior is realized by components within a system. These diagrams contain classes, associations, and their message exchanges within a collaboration to accomplish a purpose. They may be constructed using the following technique:

1. Identify behavior whose realization and implementation is to be specified.
2. Model class roles.
3. Model association roles.
4. Model message flows.
5. Refine and elaborate as required.

Collaboration diagrams

- May exist in generic form. They describe a set of classes and associations involved in message exchange sequences, that is, a collaboration among class roles and association roles, and their interactions.

- May exist in instance form. They describe a set of objects, links, and one actual message exchange sequence consistent with the generic form, that is, a scenario among objects conforming to class roles and links conforming to association roles.
- Are types of interaction diagrams that focus on a set of classes and associations involved in realizing and implementing behavior. This is depicted as a set of interactions arranged around the involved class roles and association roles.
- Show relationships among objects within an interaction.
- Are valuable for exploring all the effects of one object upon another and for exploring detailed procedural behavior.
- Must have class roles that are relevant to realizing the behavior. This includes indirectly affected or accessed transient classes (including method arguments and local variables).
- Must have association roles that are relevant to realizing the behavior. This includes transient associations among classes (including links to method arguments and local variables).
- May have message flows attached to association roles. If messages are not used on the diagram, the diagram depicts the context in which interactions can occur, but no interactions are explicitly shown.
- May have the various expressions (message flows, guard conditions, iterations, recurrence clauses, and message signatures) they utilize be expressed using pseudocode or another language.

Collaborations

Collaborations are classes that define a set of classes and associations meaningful for a set of purposes. The identification of participants and their relationships need not have global meaning; that is, the participants may be related only for the purpose of the collaboration. Collaborations are used to model a set of entities and relationships that are capable of interacting for a given set of purposes. Collaborations

- Define a context for interactions. A *context* is a set of related class roles and association roles that are used for a particular purpose. Contexts are not fully shown in sequence diagrams, since they do not show association roles; however, they are fully shown in collaboration diagrams.
- May be associated with use cases (Chapter 8). They *realize* use cases by describing the context in which behavior occurs and the interactions that realize the behavior.
- May be associated with operations. They describe the context in which operations occur. Such collaborations
  - Must have refinement relationships to the behavior they implement.
  - Must include the target class role of the operation. This class role represents the object on which the operation is applied.

  - Must include any other class roles that are called on to provide services within the execution of the behavior.
  - Must include roles for objects and links that exist before the behavior and after the behavior.
  - Must include roles for objects and links that exist only during the behavior.
  - May include objects and links that are created during an interaction. These have the keyword "new" as a constraint.
  - May include objects and links that are destroyed during an interaction. These have the keyword "destroyed" as a constraint.
  - May include objects and links that persist throughout the interaction. These have the keywords "transient" or "new destroyed" as a constraint.
  - May have message flows that are superimposed on association roles. These correspond to steps within the behavior.
- May be associated with classes to describe their static structure. The union of all the collaborations for all the operations of a class establishes an overall collaboration for the entire class. This union shows the context for the implementation of the class.
- May be used for different interactions and purposes using different sets of messages.
- May be organized hierarchically to realize a system. This is explicitly how use cases are organized.
- May have a set of constraining model elements. These are constraints expressed using classes and associations. They express extra constraints on the participating elements.
- Are anonymous when explicitly attached to named model elements.
- May be parameterized. Parameterized collaborations are also known as *template collaborations* or *patterns* (Figure 10-2). Patterns
  - Are denoted as dashed ellipses connected to classes and associations or to objects and links using dashed lines. The attached classes and associations or objects and links participate in the pattern as specified by their role names.
  - Are constructs defining a family of collaborations having a common form. The differences among collaborations in the family are represented by class and association roles, allowing a single template to instantiate many collaborations. Specific collaborations within this family of prototypes are defined by specifying actual objects and links that are bound to formal class and association roles.
  - May have a name or identifier string that represents the name of the pattern. When omitted, the collaboration is no longer freestanding and must be attached to a model element.
  - Must specify the role names to which model elements may bind. These must correspond to names of elements within the context of the

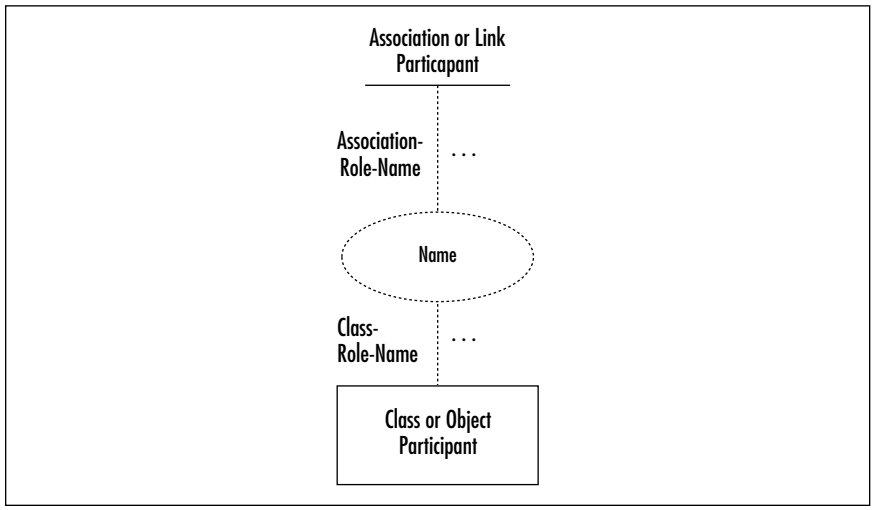


Figure 10-2: Pattern

collaboration. They are treated as parameters that are bound to specific elements on each occurrence or use of the pattern.

– Must be bound to particular model elements in order to be used. The model elements must be capable of supporting the requirements of the roles to which they bind in order to participate in the collaboration.

Association Roles

Association roles (Figure 10-3) are classes that define roles or specific parts played by association participants in collaborations. They are used to model roles that relationships may play within collaborations.

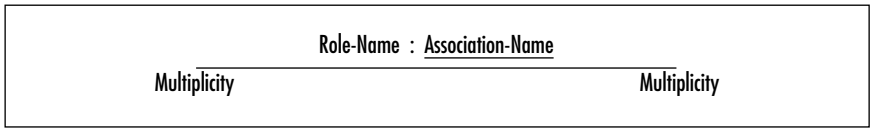


Figure 10-3: Association Role

Association roles

- Are denoted as association paths.
- Specify the types of links that may play particular roles or participate within collaborations.
- Specify a restricted view of associations. That is, they define what is required of an association for its participation in the collaboration.
- May use other notation of associations and links.

- Represent roles that bind to actual links when collaborations are used.
- Have association role ends that connect association role paths to two or more other class roles. Association role ends may have properties (Chapter 15) enclosed in curly brackets near the associated class roles.
- May have a role name or identifier string that represents the name of the role. The name must not be underlined. When omitted, the colon remains and the role is played by an anonymous link.
- May have a colon followed by an association name or identifier string that represents the association class of the role. The name must be underlined.
- Must have a multiplicity expression when connected to class roles representing sets of objects.

Multi-roles

Multi-roles (Figure 10-4) are class roles or association roles representing a set of objects or links that participate in interactions or collaborations. They are used to model roles that a set of entities plays within interactions or collaborations.

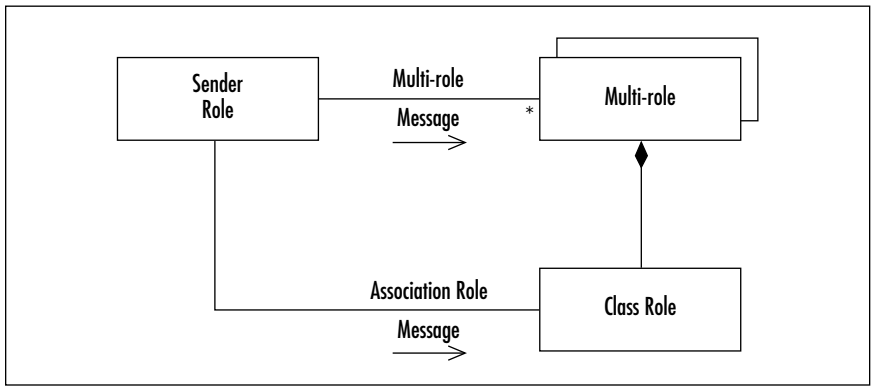


Figure 10-4: Multi-role

Multi-roles

- Are denoted as two rectangles in which the top rectangle is shifted slightly vertically and horizontally to suggest a stack of rectangles.
- May use other notation of classes, objects, associations, and links.
- Require two messages to perform an operation on each object in a set of instances. This includes one message to extract links to individual objects, and one message to each individual object using the extracted link. The two mentioned messages may be combined into a single message that includes an iteration and an application of an operation to each individual object.
- As class roles, they
  - Appear as two rectangles in which the top rectangle is shifted slightly vertically and horizontally to suggest a stack of rectangles.

  - May receive messages. This indicates that a message is received by the set of instances and not each instance in the set. This use of multi-roles may include a selection operation to find an individual object. Such messages return a reference to an individual object with which the sender may interact. Messages to single class roles indicate a message to an individual object.
  - May be attached to the multiclass role using a composition link.
- As association roles, they
  - Must use the many multiplicity indicator (*) attached to multiclass roles to show that many individual links are implied.
  - May propagate messages.

Message Flows

Message flows (Figure 10-5) involve the sending of messages between class roles via association roles, including a method call, a signal sent between active objects, or an explicit raising of an event. Message flows correspond to messages sent between class roles in sequence diagrams.

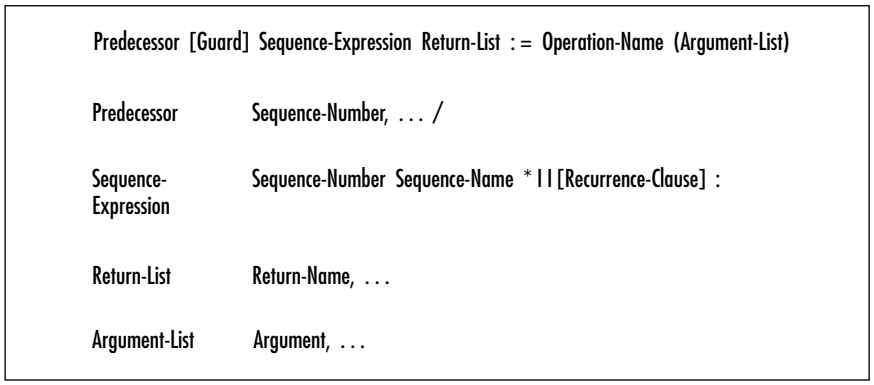


Figure 10-5: Message Flow

Message flows

- Are denoted as labeled arrows attached to association roles between class roles. The sender will send the message, and the receiver will receive the message. The link is used to transport or otherwise implement the delivery of the message.
- May have multiple messages attached to the same arrow.
- May have a predecessor specification consisting of a comma-separated list of sequence numbers that must have occurred (and completed) in order to enable the message flow. Each sequence number must match the sequence number of another message. Numerically preceding sequence numbers are implicit predecessors and need not be explicitly listed. Predecessor specification is used for the synchronization of threads.

- May have square brackets containing a guard condition. This is a Boolean condition that must be satisfied in order to enable the message flow.
- May have a sequence expression. This is an expression involving nesting, concurrency, branching, and iteration control. Sequence expressions
  - Must have a sequence number. This is an integer that indicates the sequence of the messages within the next higher level of nesting. It starts at 1 and increments by 1. Messages that differ in one integer are sequentially related at the same level of nesting when associated with the same class role; that is, within activation 1, message 1.1 precedes message 1.2, message 1.2 precedes message 1.3, and so forth. This represents the nesting of control via the dot notation. If control is concurrent, nesting does not occur; that is, messages 1 and 2 are concurrent if these messages are associated with different class roles. Other numbering schemes may also be used, but they must be defined in the context in which they are used.
  - May have a sequence name. This is a string representing the thread of control. Messages that differ in the final name are concurrent at the same level of nesting. Within activation 1, message 1.1a and message 1.1b are concurrent via threads a and b. All threads of control are equal within a nesting level. This represents concurrency.
  - May have a recurrence clause.

    An asterisk indicates that the recurrence clause is an iteration specification in which messages are executed sequentially at the nesting depth. An asterisk followed by a double vertical line indicates that the recurrence clause is an iteration specification in which messages are executed concurrently. The clause specifies the number of times the message is sent. An asterisk (with or without a double vertical line) without a specified iteration (and without square brackets) indicates that the message is repeatedly sent an unspecified number of times. When the recurrence clause is omitted, the asterisk, double vertical lines, and square brackets are omitted.

    Square brackets alone indicate that the recurrence clause is a condition specification. This specifies that the execution of the message is contingent on the truth of the condition. When the clause is omitted, the square brackets are omitted.

    A loop involving multiple messages at the same level of nesting may be specified by placing an asterisk and a recurrence clause in a qualifier attached to the class role and attaching all messages within the loop as originating from the qualifier. Alternatively, the loop may be specified by placing an asterisk and a recurrence clause in a constraint expression (Chapter 15) that is attached to the messages within the iteration.
  - Are concatenated together in a dot-separated list at each level of nesting. Each nesting level may have sequence numbers and sequence names concatenated. However, recurrence clauses are not concatenated since each nesting level specifies its own recurrence clause within the enclosing context.
- May have a return list consisting of a comma-separated list of names that designate the values returned by the operation. These values may be used as

actual parameters in subsequent messages. The return names must match the order, number, and types specified in the formal return list of the operation. When the list is omitted, the assignment operator (:=) is omitted.

- Must have a name or identifier string that represents the name of the operation. The operation must be in the class role receiving the message.
- May have parentheses containing an argument list consisting of a comma-separated list of actual parameters passed to a method. The actual parameters must match the order, number, and types specified in the formal parameter list of the operation.

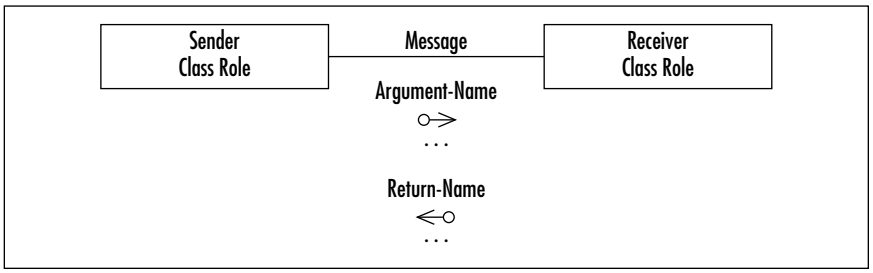


Figure 10-6: Data Tokens

- May express argument lists and return lists using data tokens (Figure 10-6). Data tokens
  - Represent argument names or values and return names.
  - Are small circles shown near messages.
  - Are labeled with argument names or return names.
  - Have a small arrow pointing in the direction of the movement of data. For argument names, they point in the direction of the message. For return names, they point in the opposite direction from the message.

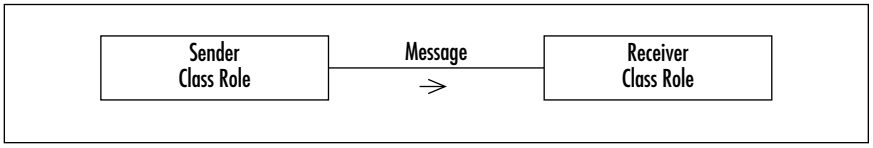


Figure 10-7: Simple Message Flow

- May be simple messages (Figure 10-7).
  - Are denoted as stick arrowheads.
  - This type of message flow represents a flat flow of control.
  - This type of message flow is depicted using a stick arrowhead.
  - Control is passed from the sender to the receiver.
  - No other details about the communication are provided, either because they are not known or are not relevant.
- May be synchronous messages (Figure 10-8).
  - Are denoted as filled solid arrowheads.

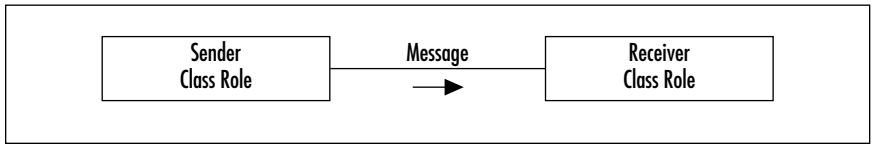


Figure 10-8: Synchronous Message Flow

- This type of message flow represents a nested flow of control via an operation call. The operation call invokes an operation synchronously.
- This type of message flow is depicted using a filled arrowhead.
- The sender passes control to the receiver via the message and pauses to wait for the receiver to relinquish or return control.
- The receiver performs the operation, may pass control to other class roles if necessary, and returns control to the caller once the operation is completed. The receiver may also return or send information back to the sender.
- Nested sequences complete before their outer sequences resume.
- The receiver must be a passive object.

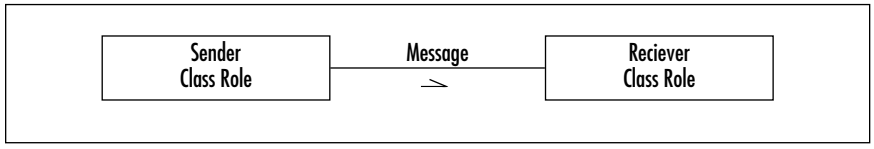


Figure 10-9: Asynchronous Message Flow

- May be asynchronous messages (Figure 10-9).
  - Are denoted as half stick arrowheads.
  - This type of message flow represents a nonnested flow of control via a signal. The signal invokes an operation asynchronously.
  - This type of message flow is depicted using a half-arrowhead.
  - The sender signals the receiver via the message and continues with the sequence without pausing to wait for the receiver.
  - The receiver performs the operation and may return or send information back to the sender.
  - The sender and receiver sequences are concurrent.
  - The receiver must be an active object.
  - If the receiver returns information to the sender, the information must be shown explicitly.

,ch10.18470 Page 186 Wednesday, May 9, 2001 4:04 PM