15.3 Common UML Interaction Diagram Notation

Illustrating Participants with Lifeline Boxes

In the UML, the boxes you've seen in the prior sample interaction diagrams are called lifeline boxes. Their precise UML definition is subtle, but informally they represent the participants in the interaction—related parts defined in the context of some structure diagram, such as a class diagram. It is not precisely accurate to say that a lifeline box equals an instance of a class, but informally and practically, the participants will often be interpreted as such. Therefore, in this text I'll often write something like "the lifeline representing a Sale instance," as a convenient shorthand. See Figure 15.5 for common cases of notation.

![Lifeline boxes to show participants in interactions.](image)

Figure 15.5 Lifeline boxes to show participants in interactions.

15.4 Basic Sequence Diagram Notation

Lifeline Boxes and Lifelines

In contrast to communication diagrams, in sequence diagrams the lifeline boxes include a vertical line extending below them—these are the actual lifelines. Although virtually all UML examples show the lifeline as dashed (because of UML 1 influence), in fact the UML 2 specification says it may be solid or dashed.
Messages

Each (typical synchronous) message between objects is represented with a message expression on a filled-arrowed solid line between the vertical lifelines (see Figure 15.7). The time ordering is organized from top to bottom of lifelines.

Figure 15.7 Messages and focus of control with execution specification bar.

In the example of Figure 15.7 the starting message is called a found message in the UML, shown with an opening solid ball; it implies the sender will not be specified, is not known, or that the message is coming from a random source. However, by convention a team or tool may ignore showing this, and instead use a regular message line without the ball, intending by convention it is a found message.4

Focus of Control and Execution Specification Bars

As illustrated in Figure 15.7, sequence diagrams may also show the focus of control (informally, in a regular blocking call, the operation is on the call stack) using an execution specification bar (previously called an activation bar or simply an activation in UML 1). The bar is optional.

Guideline: Drawing the bar is more common (and often automatic) when using a UML CASE tool, and less common when wall sketching.

---

3. An open message arrow means an asynchronous message in an interaction diagram.
4. Therefore, many of the book examples won’t bother with the found message notation.
Illustrating Reply or Returns

There are two ways to show the return result from a message:

1. Using the message syntax `returnVar = message(parameter).
2. Using a reply (or return) message line at the end of an activation bar.

Both are common in practice. I prefer the first approach when sketching, as it's less effort. If the reply line is used, the line is normally labelled with an arbitrary description of the returning value. See Figure 15.8.

![Figure 15.8](image)

Figure 15.8 Two ways to show a return result from a message.

Messages to "self" or "this"

You can show a message being sent from an object to itself by using a nested activation bar (see Figure 15.9).

![Figure 15.9](image)

Figure 15.9 Messages to "this."

Creation of Instances

Object creation notation is shown in Figure 15.10. Note the UML-mandated dashed line.
The typical interpretation (in languages such as Java or C#) of a create message on a dashed line with a filled arrow is "invoke the new operator and call the constructor".

*Figure 15.10* Instance creation and object lifelines.

**Object Lifelines and Object Destruction**

In some circumstances it is desirable to show explicit destruction of an object. For example, when using C++ which does not have automatic garbage collection, or when you want to especially indicate an object is no longer usable (such as a closed database connection). The UML lifeline notation provides a way to express this destruction (see Figure 15.11).

*Figure 15.11* Object destruction.

**Diagram Frames in UML Sequence Diagrams**

To support conditional and looping constructs (among many other things), the UML uses *frames*. Frames are regions or fragments of the diagrams; they have
an operator or label (such as loop) and a guard\(^7\) (conditional clause). See Figure 15.12.

![Diagram of UML frame]

Figure 15.12 Example UML frame.

The following table summarizes some common frame operators:

<table>
<thead>
<tr>
<th>Frame Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>alt</td>
<td>Alternative fragment for mutual exclusion conditional logic expressed in the guards.</td>
</tr>
<tr>
<td>loop</td>
<td>Loop fragment while guard is true. Can also write (\text{loop}(n)) to indicate looping (n) times. There is discussion that the specification will be enhanced to define a (\text{FOR}) loop, such as (\text{loop}(i, 1, 10))</td>
</tr>
<tr>
<td>opt</td>
<td>Optional fragment that executes if guard is true.</td>
</tr>
</tbody>
</table>

**Looping**

The LOOP frame notation to show looping is shown in Figure 15.12.

**Conditional Messages**

An OPT frame is placed around one or more messages. Notice that the guard is

---

6. Also called diagram frames or interaction frames.
7. The \(\text{boolean test}\) guard should be placed over the lifeline to which it belongs.
placed over the related lifeline. See Figure 15.13.

**Figure 15.13** A conditional message.

*Conditional Messages in UML 1.x Style—Still Useful?*

The UML 2.x notation to show a single conditional message is heavyweight, requiring an entire OPT frame box around one message (see Figure 15.13). The older UML 1.x notation for single conditional messages in sequence diagrams is not legal in UML 2, but so simple that especially when sketching it will probably be popular for years to come. See Figure 15.14.

**Figure 15.14** A conditional message in UML 1.x notation—a simple style.

**Guideline:** Use UML 1 style only for simple single messages when sketching.

*Mutually Exclusive Conditional Messages*

An ALT frame is placed around the mutually exclusive alternatives. See Figure 15.15.
**Iteration Over a Collection**

A common algorithm is to iterate over all members of a collection (such as a list or map), sending the same message to each. Often, some kind of iterator object is ultimately used, such as an implementation of `java.util.Iterator` or a C++ standard library iterator, although in the sequence diagram that low-level "mechanism" need not be shown in the interest of brevity or abstraction.

At the time of this writing, the UML specification did not (and may never) have an official idiom for this case. Two alternatives are shown—reviewed with the leader of the UML 2 interaction specification—in Figure 15.16 and Figure 15.17.

---

**Figure 15.15** Mutually exclusive conditional messages.

**Figure 15.16** Iteration over a collection using relatively explicit notation.
Note the **selector** expression `lineItems[i]` in the lifeline of Figure 15.16. The selector expression is used to select one object from a group. Lifeline participants should represent one object, not a collection.

In Java, for example, the following code listing is a possible implementation that maps the explicit use of the incrementing variable `i` in Figure 15.16 to an idiomatic solution in Java, using its enhanced `for` statement (C# has the same).

```java
public class Sale {
    private List<SalesLineItem> lineItems = new ArrayList<SalesLineItem>();

    public Money getTotal() {
        Money total = new Money();
        Money subtotal = null;
        for (SalesLineItem lineItem : lineItems) {
            subtotal = lineItem.getSubtotal();
            total.add( subtotal );
        }
        return total;
    }
    // ...
}
```

Another variation is shown in Figure 15.17; the intent is the same, but details are excluded. A team or tool could agree on this simple style by convention to imply iteration over all the collection elements.8

![Diagram](image)

**Figure 15.17** Iteration over a collection leaving things more implicit.

---

8. I use this style later in the book.
Nesting of Frames

Frames can be nested. See Figure 15.18.

![Diagram of nested frames](image)

Figure 15.18 Nesting of frames.

How to Relate Interaction Diagrams?

Figure 15.19 illustrates probably better than words. An interaction occurrence (also called an interaction use) is a reference to an interaction within another interaction. It is useful, for example, when you want to simplify a diagram and factor out a portion into another diagram, or there is a reusable interaction occurrence. UML tools take advantage of them, because of their usefulness in relating and linking diagrams.

They are created with two related frames:

- a frame around an entire sequence diagram, labeled with the tag sd and a name, such as AuthenticateUser

- a frame tagged ref, called a reference, that refers to another named sequence diagram; it is the actual interaction occurrence

Interaction overview diagrams also contain a set of reference frames (interaction occurrences). These diagrams organized references into a larger structure of logic and process flow.

**Guideline:** Any sequence diagram can be surrounded with an sd frame and name it. Frame and name one when you want to refer to it using a ref frame.

![Guideline diagram](image)

Figure 15.19 Example interaction occurrence, sd and ref frames.