

Essentials of interaction diagrams - Chapter 9 of text

Essentials of interaction diagrams - Chapter 9 of text	1
• A few general points	2
• Collaborations	2
• Collaboration diagrams	3
• Activation	3
• Sequence diagrams	6
Three versions of a sequence diagram illustrating various features	6
Recap on main elements of sequence diagrams	7
Other aspects of sequence diagrams	7
• Additional points on interaction diagrams et al	8
* Suppressing detailed behaviour	8
* Returned values; Creation and deletion of objects (diagrams below)	8
* Good practice: Law of Demeter Only talk to immediate friends"!	9
* (A few) Miscellaneous points	9

• A few general points

- Text points out that the *use case* and *class* models are most important – *still we do need to get into enough detailed analysis early on to get a proper understanding of all key issues.*
- In the introductory case study we had an example of a *sequence diagram* being used to show how objects (in a class model) interacted to carry out a specific use case.
- In this section we look a little more closely at sequence diagrams and also at *collaboration diagrams* (the second kind of interaction diagram)
- Text notes that one may not need to have interaction diagrams for all use cases. Also, the level of detail to include is a matter of judgement (*but see earlier comment*)
- Normal to revise class model while developing interaction diagrams but make sure overall class model remains well-structured.
- We start by saying what is meant by a "*collaboration*".

• Collaborations

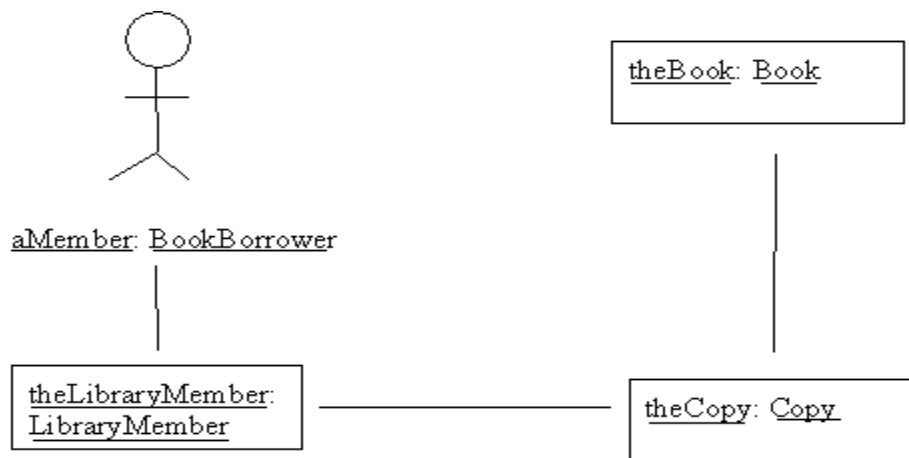


Figure 9.1 (of text): A simple collaboration, showing no interaction

- As illustrated above, a collaboration consists of objects, actors and links. Can be thought of as like an instance of part of a class model. Only relevant elements (*e.g. to a given use case*) need be shown.
- Actor that starts the use case is called the initiator [*only one actor above*]

• Collaboration diagrams

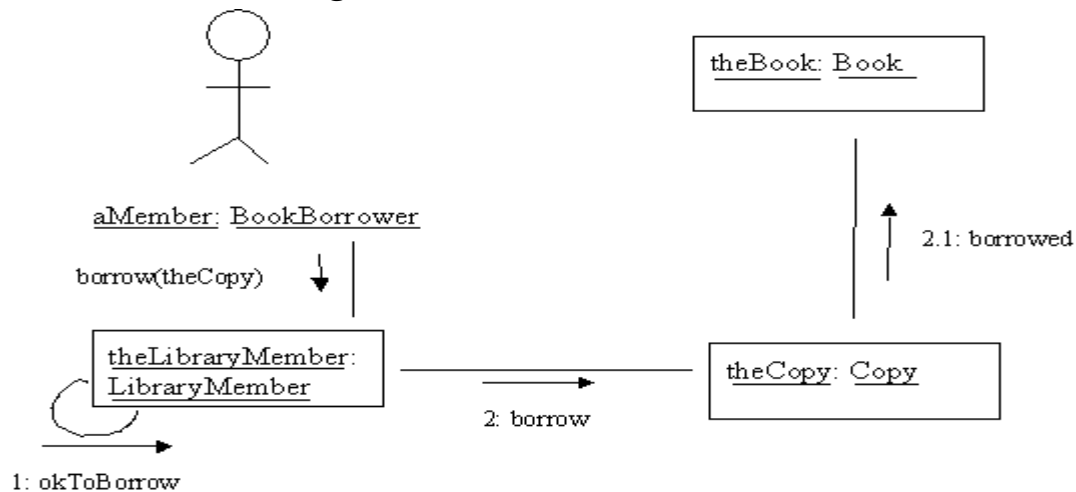


Figure 9.2 (of text): Interaction shown on a collaboration diagram

- Type of diagram is good for *showing links between objects*
- New elements (compared to Fig 9.1) are *messages*
- Labeled arrows show messages sent *from* object at tail *to* object at head (called the target object)
- Developing interaction diagrams can help to identify needed associations between classes and operations within classes - *e.g. the target object must provide the appropriate operation [method if Java] to respond to a message.*

• Activation

- The text raises the important idea of *activation*, which we explain in the following.
Note: text uses the term *procedural* to distinguish non-concurrent interactions. In this sense, procedural means only one object is computing at a time.
- We say that an object starts to compute when it is *activated* by receiving a message.
 E.G. consider the following (simple) sequence of events for objects A, B, C and D:

Event	Object with control	Live activation stack
Message received by A (from external source)	A	Empty
Message (1) received by B from A	B	A
Message (1.1) received by C from B	C	B A
Message (1.1.1) received by D from C	D	C B A
Message reply received by C from D	C	B A
Message reply received by B from C	B	A
Message reply received by A from B	A	Empty
A finishes computing	None	Empty

-- The above illustrates *flow of control* from one object to another.

-- May be helpful to think of *control* as a token that gets passed as a message along the links in a collaboration diagram.

-- When considering non-concurrent interactions, only actors can initiate activity. On the other hand, where concurrency is possible, one can have so-called *active objects* that can also initiate activity. (recall the clock in our simple watch example)

-- UML *message numbering system* is illustrated in above example. More generally, the system is

<p>--- Initial message from actor to an object is not numbered</p> <p>--- First message from one object to another is number 1</p> <p>--- Whenever object O receives a message, the number of that message will be used as a prefix of all messages that are sent until O replies to that message.</p>
--

• **Sequence diagrams**

Three versions of a sequence diagram illustrating various features

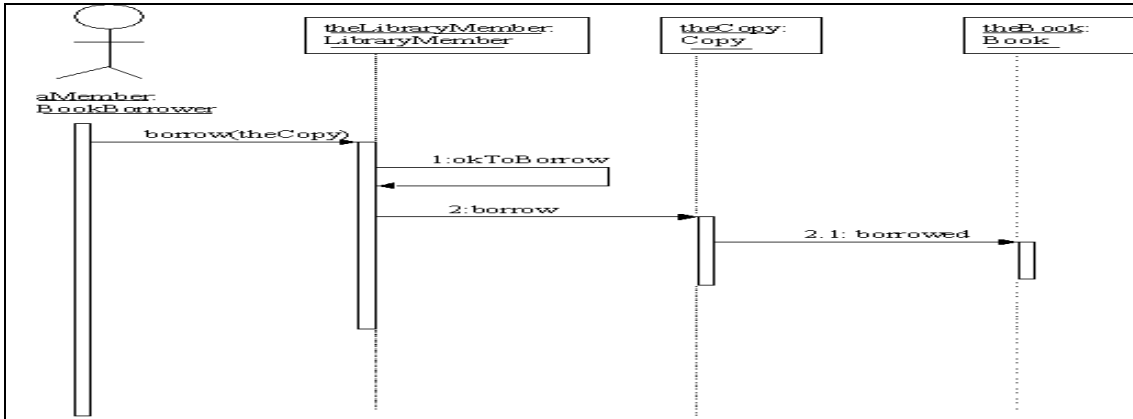


Fig 9.3 (of text): Interaction shown on a sequence diagram

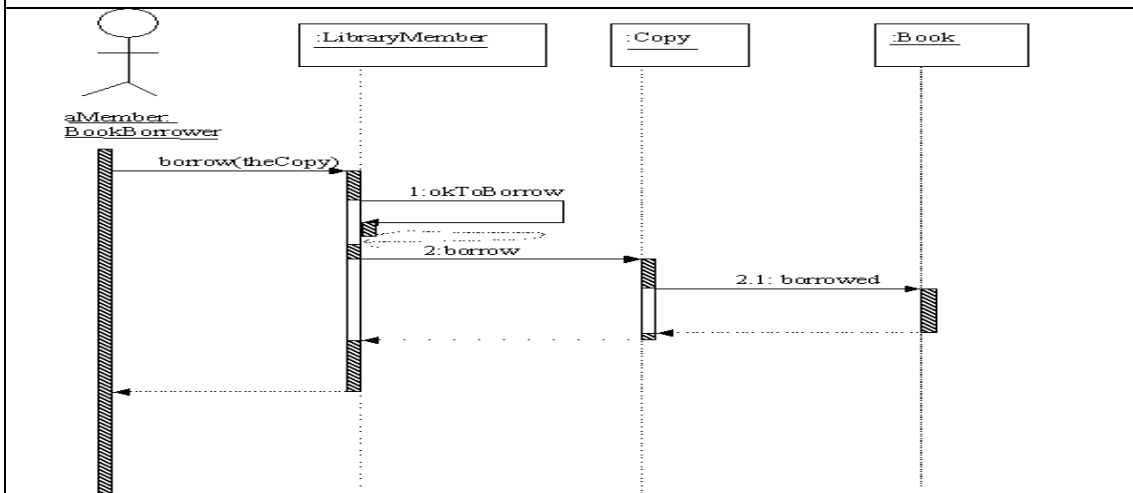


Fig 9.5 (of text): Interaction shown on a sequence diagram, with optional features)

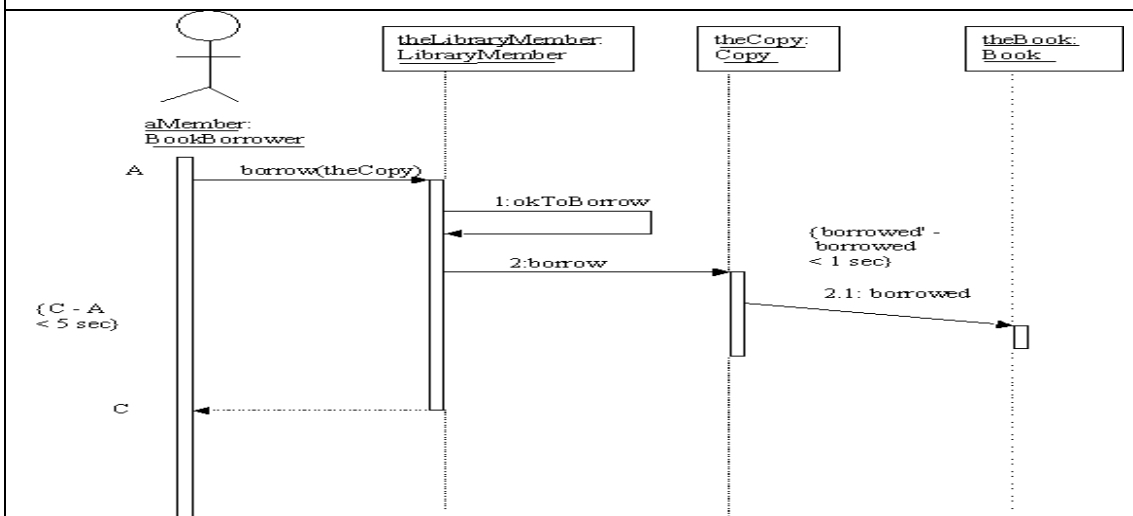


Fig 9.9 (of text): Showing time constraints on a sequence diagram

Recap on main elements of sequence diagrams

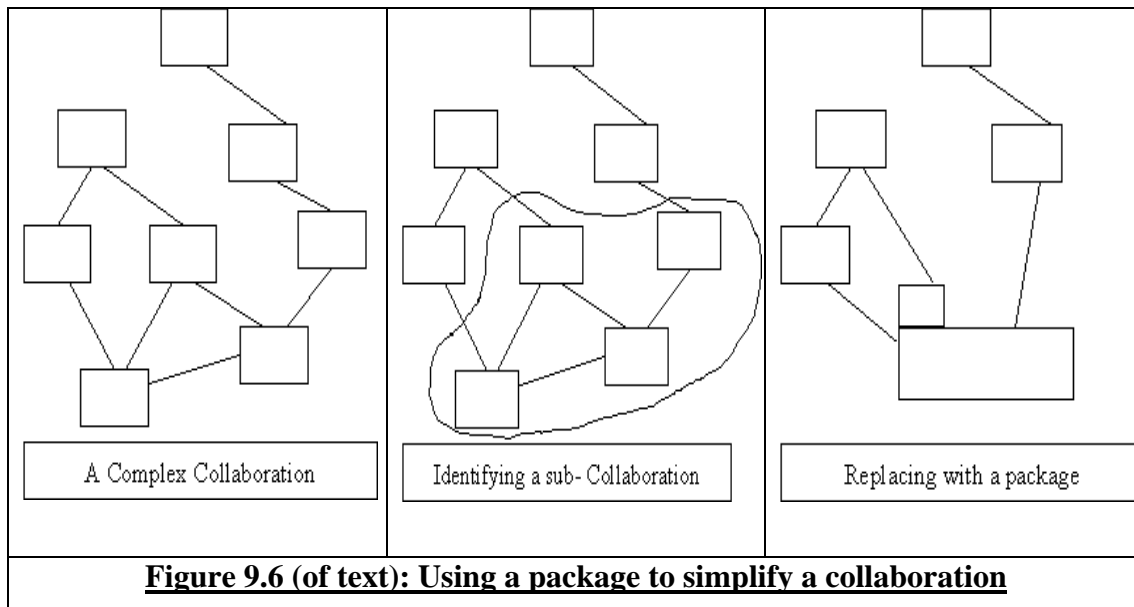
- Fig 9.3 shows which messages are passed between objects and in what order they must occur - read the messages from top to bottom
- Fig 9.3 shows what happens when a library member borrows a copy of the book. It starts from the position of having a certain object of class "LibraryMember" called "theLibraryMember" and a certain object of class "Copy" called "theCopy" (corresponding to a person bringing a physical copy to the issue desk to borrow it)
- "theLibraryMember" acts (as decided earlier for this example) on behalf of the real library member so interaction begins with a message "borrow(theCopy)" passed to it
- Then, after a check if allowed to borrow, object "theLibraryMember" sends the message "borrow" to "theCopy". Finally, "theCopy" sends a message "borrowed" to "theBook" - we need to update system information on how many copies are on loan.

Other aspects of sequence diagrams

- In principle, the order of objects does not matter. However, for (human) readability, try to arrange that most messages flow from left to right.
- Fig 9.5 illustrates optional shading to highlight when an object is actually computing. Also shows explicitly (dashed arrows) when responses to messages occur.
- Complication when an object sends a message to itself - *analogous to a recursive procedure call in a programming language*. In effect, object is then associated with 2 (or more) live activations. This "nested activation" can be shown by offsetting the shading for the new activation as illustrated in Fig 9.5 - *Text has better diagram*. If a diagram is becoming too cluttered, may be better not to show some or all "internal" messages.
- Fig 9.9 illustrates two ways that times and timing constraints may be depicted. First shows how a constraint on the time it takes the system to respond to an actor's message can be specified. Second shows how a constraint on the time it takes for a message to pass between objects can be stated (sloping line to emphasise more than zero time, borrowed and borrowed' refer to times sent and received, resp.

• **Additional points on interaction diagrams et al**

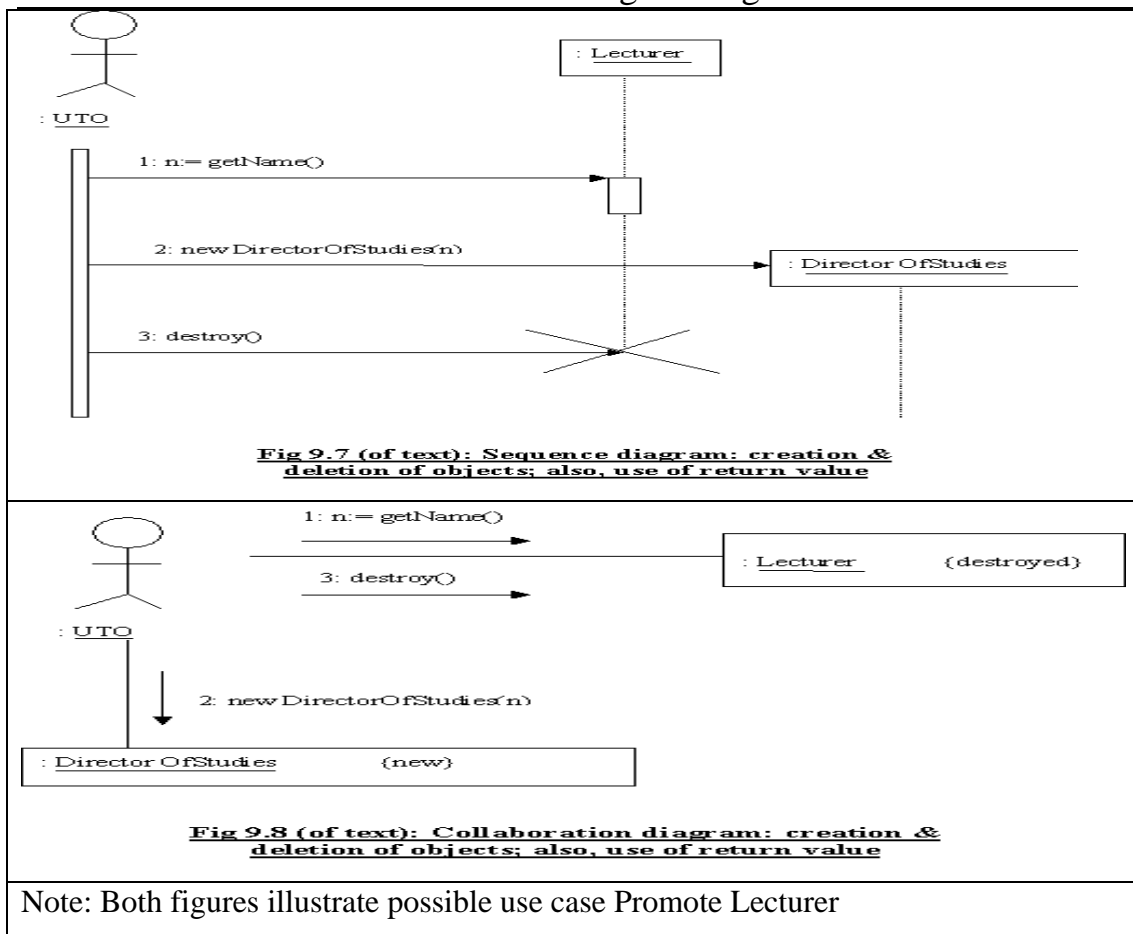
* **Suppressing detailed behaviour**



- Diagram is essentially self-explanatory; point is to suppress unnecessary detail. A "tab" is included on the package to identify it as such, and the package is given a name. (We will come back to packages, Chapter 14 of text).

* **Returned values; Creation and deletion of objects (diagrams below)**

- Assignment used to name (and store) returned value from a message
- **Collaboration:** Constraints {new}, {destroyed}, {transient} show objects created, destroyed, created & destroyed by an interaction
- **Sequence:** "Created" shown by positioning of object box (no longer at top)
Large "X" to mark end of its activation for a destroyed object. If not destroying itself, message from destroying object into the "X" (see Fig 9.7)
- Text notes that mechanisms for creating (and maybe initialising) and destroying objects vary between languages. Also, refers to the issue of "garbage collection", whether automatic or programmer responsibility, and the danger of "memory leaks" due to failure to destroy no longer needed objects. Design must specify where responsibilities for destroying objects lie.



*** Good practice: Law of Demeter "Only talk to immediate friends"!**

In response to a message m , an object O should send messages only to

- (1) O itself
- (2) objects which are sent as arguments to the message m
- (3) objects which O creates as part of its reaction to m , or
- (4) objects directly accessible from O , i.e. that use values of attributes of O .

If not followed design will be hard to maintain. Just a guideline not a real law; occasionally may be best to break it. (*text pp118, 119 for more detail*)

*** (A few) Miscellaneous points**

- Interaction diagrams can be used to show how
 - an operation is provided by an object, i.e. not just for full use cases.
 - a design pattern works (see later on possibly)
 - a complex component should be used (part of its documentation)
- Some UML ambiguities: (p. 114 whether some features are allowed on both types of interaction diagrams, p. 121 levels of granularity).