

UML (Unified Modeling Language)

7 Interactive Diagrams

Software Institute of
Nanjing University

[Interaction Diagrams]

- Interaction diagrams describe how groups of objects collaborate in some behavior. The UML defines several forms of interaction diagram, of which the most common is the sequence diagram.
 - Sequence Diagrams
 - Communication Diagrams
 - Interactive Overview Diagrams (UML2)
 - Timing Diagrams (UML2)

[1. Interaction]

- An *interaction* is a behavior that comprises a set of messages exchanged among a set of objects within a context to accomplish a purpose.

2. Sequence Diagram

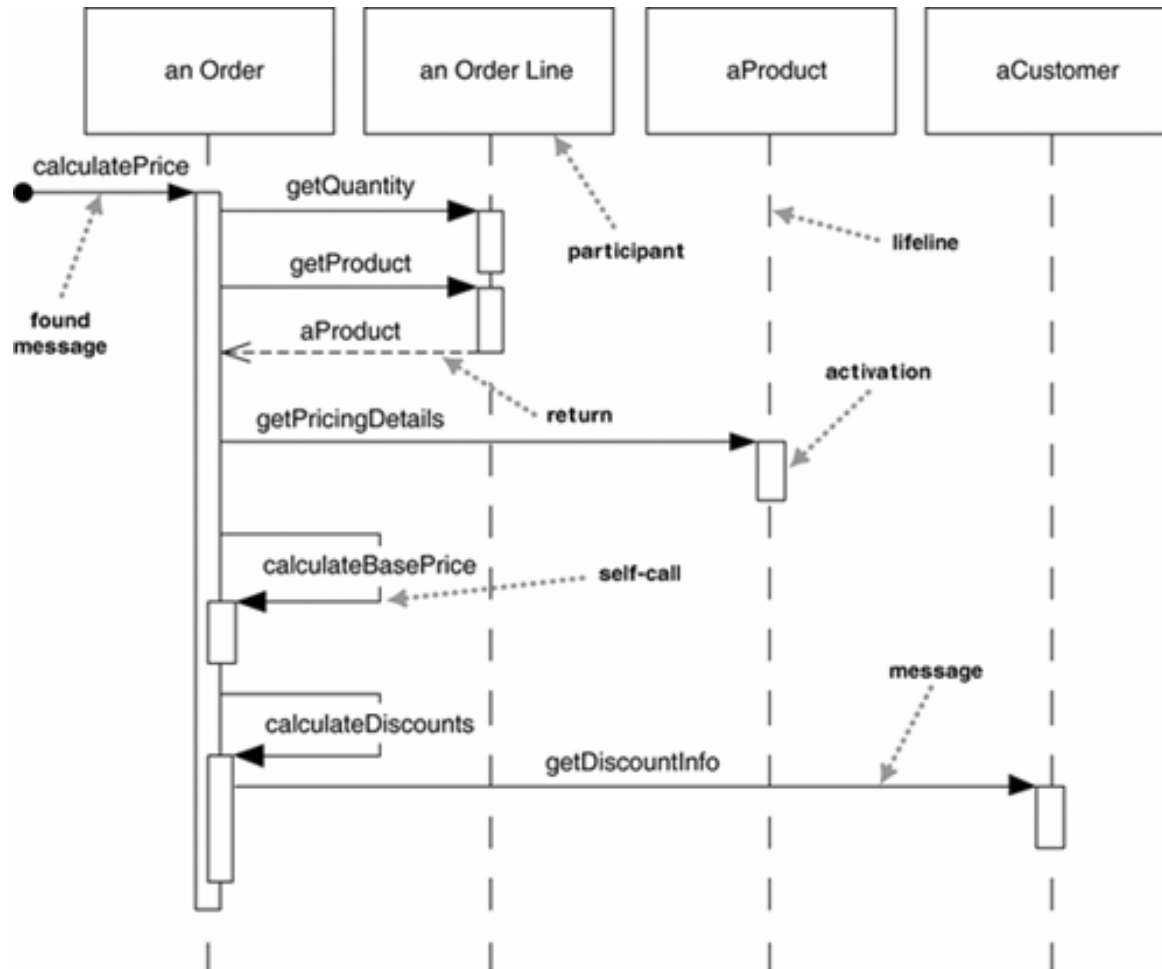
- Typically, a sequence diagram captures the behavior of a single scenario. The diagram shows a number of example objects and the messages that are passed between these objects within the use case.

2. Sequence Diagram (cont')

■ Example:

- We have an order and are going to invoke a command on it to calculate its price.
- To do that, the order needs to look at all the line items on the order and determine their prices, which are based on the pricing rules of the order line's products.
- Having done that for all the line items, the order then needs to compute an overall discount, which is based on rules tied to the customer.

2. Sequence Diagram (cont')



2.1 Links

- A **link** is a semantic connection among objects.
- A link specifies a path along which one object can dispatch a message to another (or the same) object.
- Most of the time, it is sufficient to specify that such a path exists.

[2.1 Links (cont')]

- If you need to be more precise about how that path exists, you can adorn the appropriate end of the link with any of the following standard stereotypes.
 - <<association>>
 - <<self>>
 - <<global>>
 - <<local>>
 - <<parameter>>

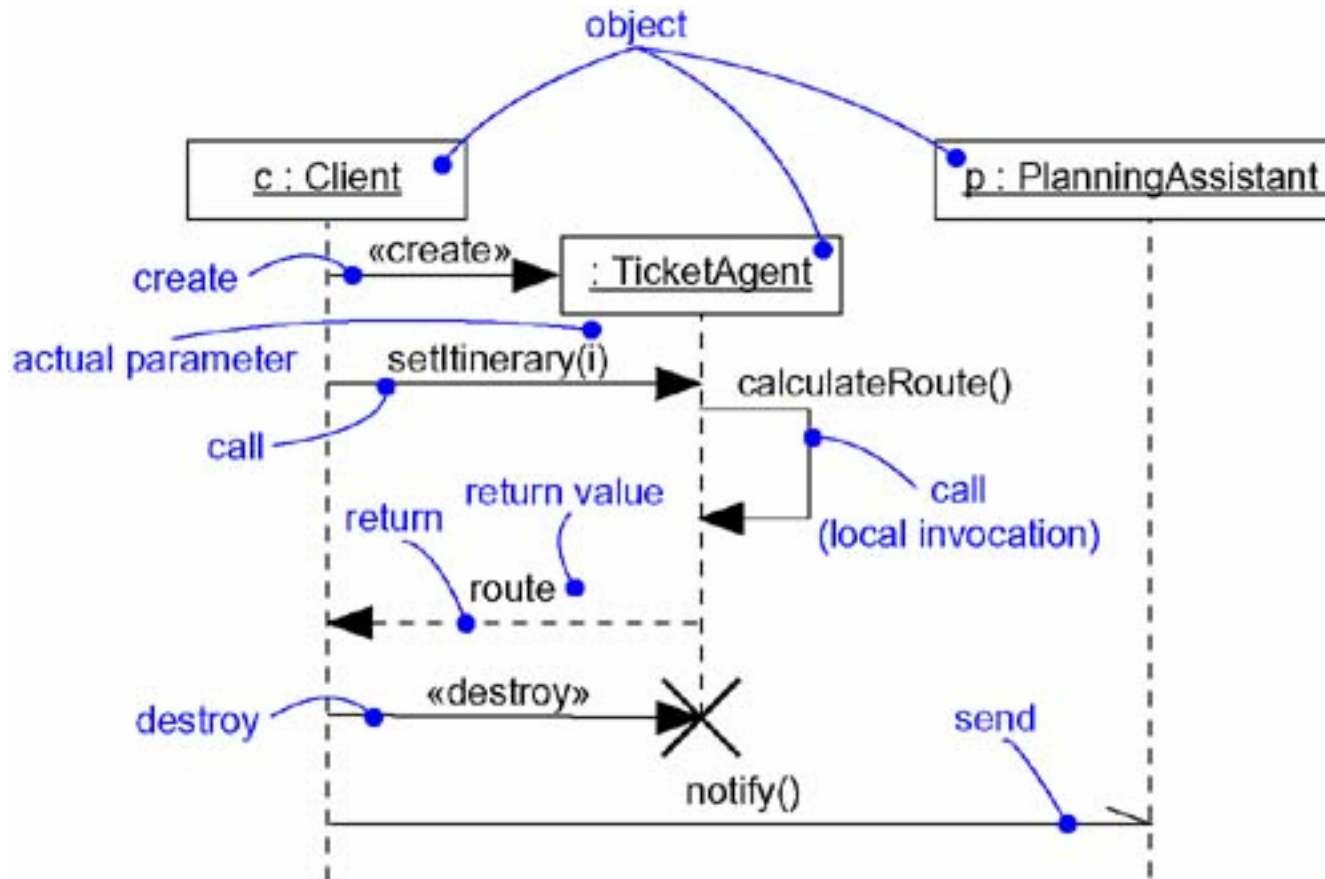
2.2 Message

- A *message* is a specification of a communication between objects that conveys information with the expectation that activity will ensue.

2.2 Message (cont')

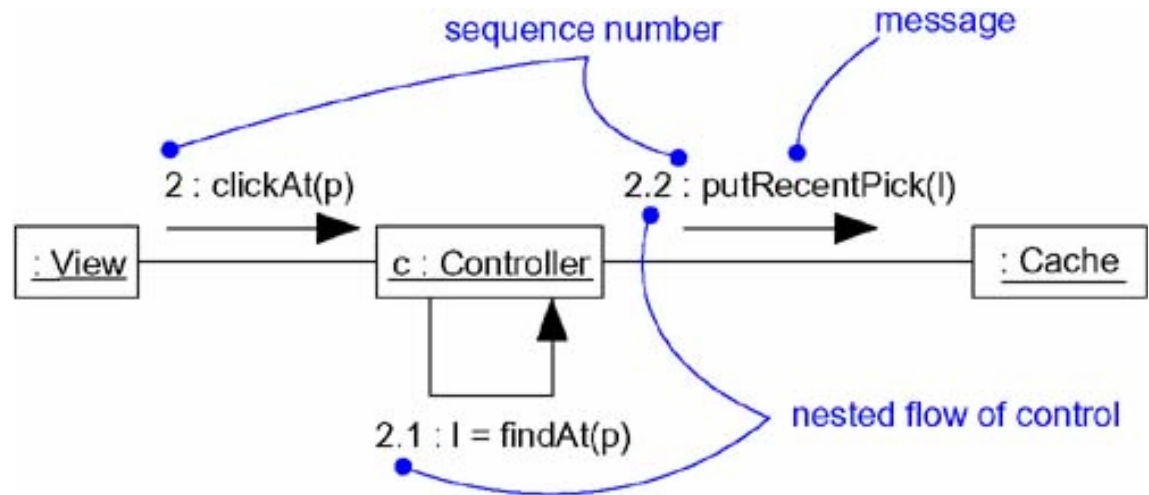
- In the UML, you can model several kinds of actions.
 - Call : Invokes an operation on an object; an object may send a message to itself, resulting in the local invocation of an operation
 - Return : Returns a value to the caller
 - Send : Sends a signal to an object
 - Create : Creates an object
 - Destroy : Destroys an object; an object may commit suicide by destroying itself

2.2 Message (cont')

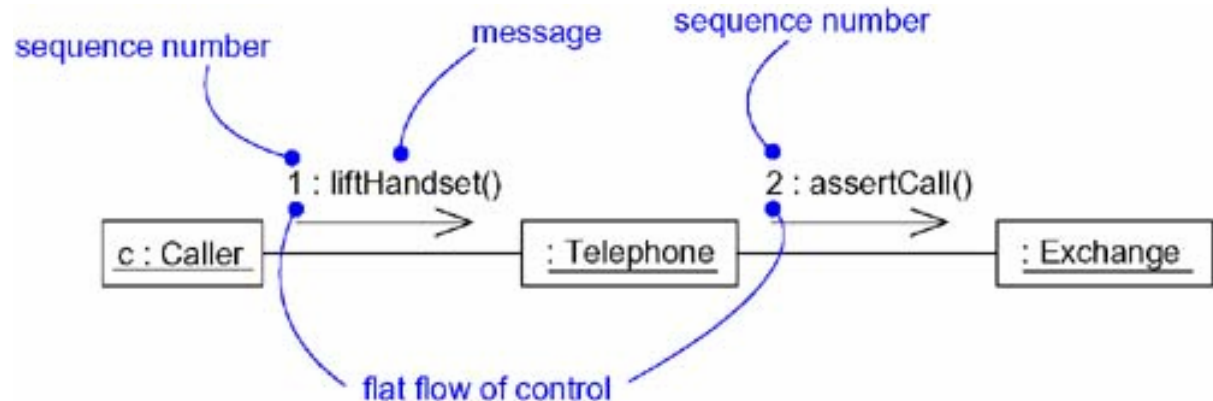


2.3 Sequencing

■ Procedural Sequence



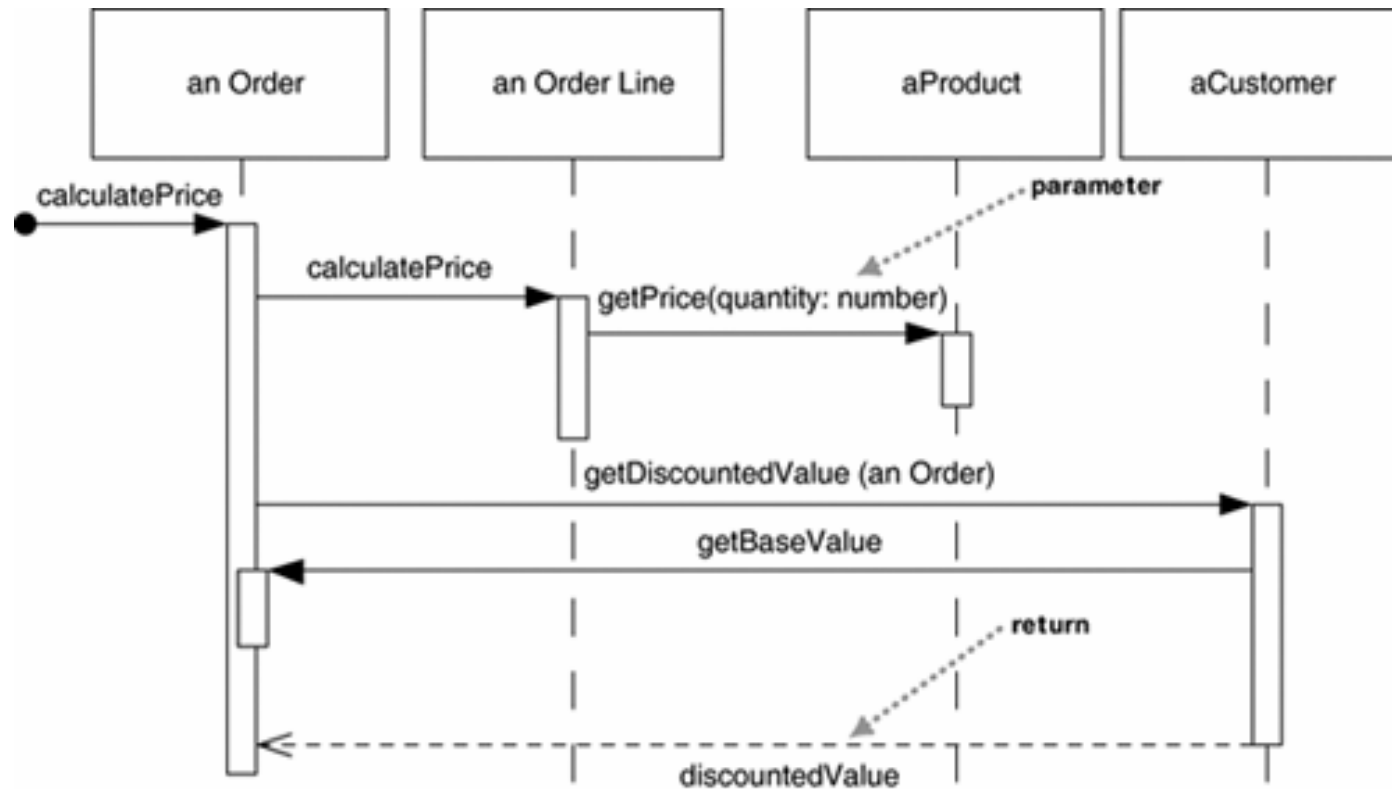
■ Flat Sequence



2. Sequence Diagram (cont')

- For another approach to this scenario.
- The Order asks each Order Line to calculate its own Price.
- The Order Line itself further hands off the calculation to the Product.
- Similarly, to calculate the discount, the Order invokes a method on the Customer. Because it needs information from the Order to do this, the Customer makes a reentrant call (getBaseValue) to the Order to get the data.

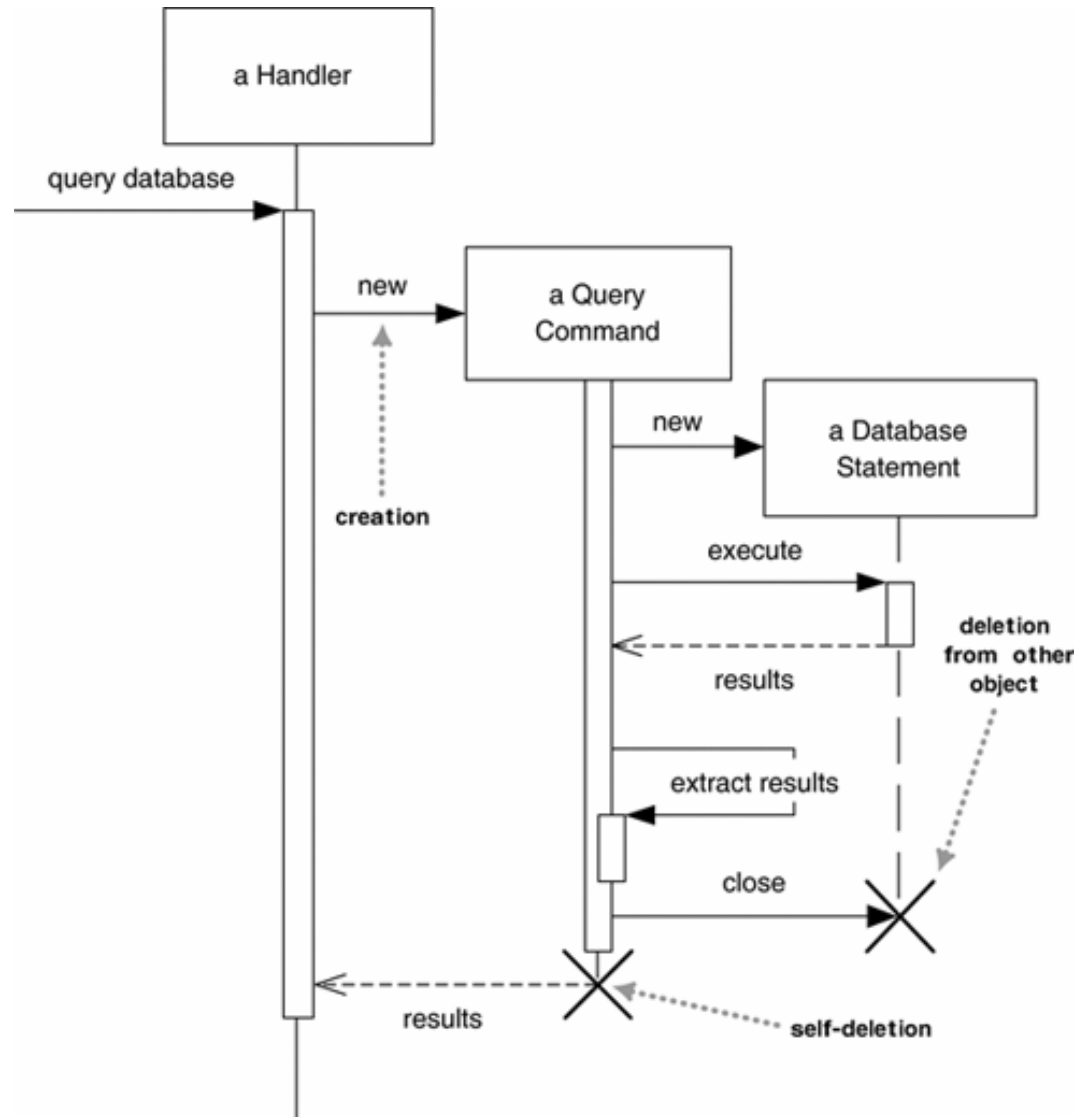
2. Sequence Diagram (cont')



2. Sequence Diagram (cont')

- The sequence diagram indicates the differences in how the participants interact.
- No1 : **centralized control**
 - with one participant pretty much doing all the processing and other participants there to supply data.
- No2 : **distributed control**
 - in which the processing is split among many participants, each one doing a little bit of the algorithm

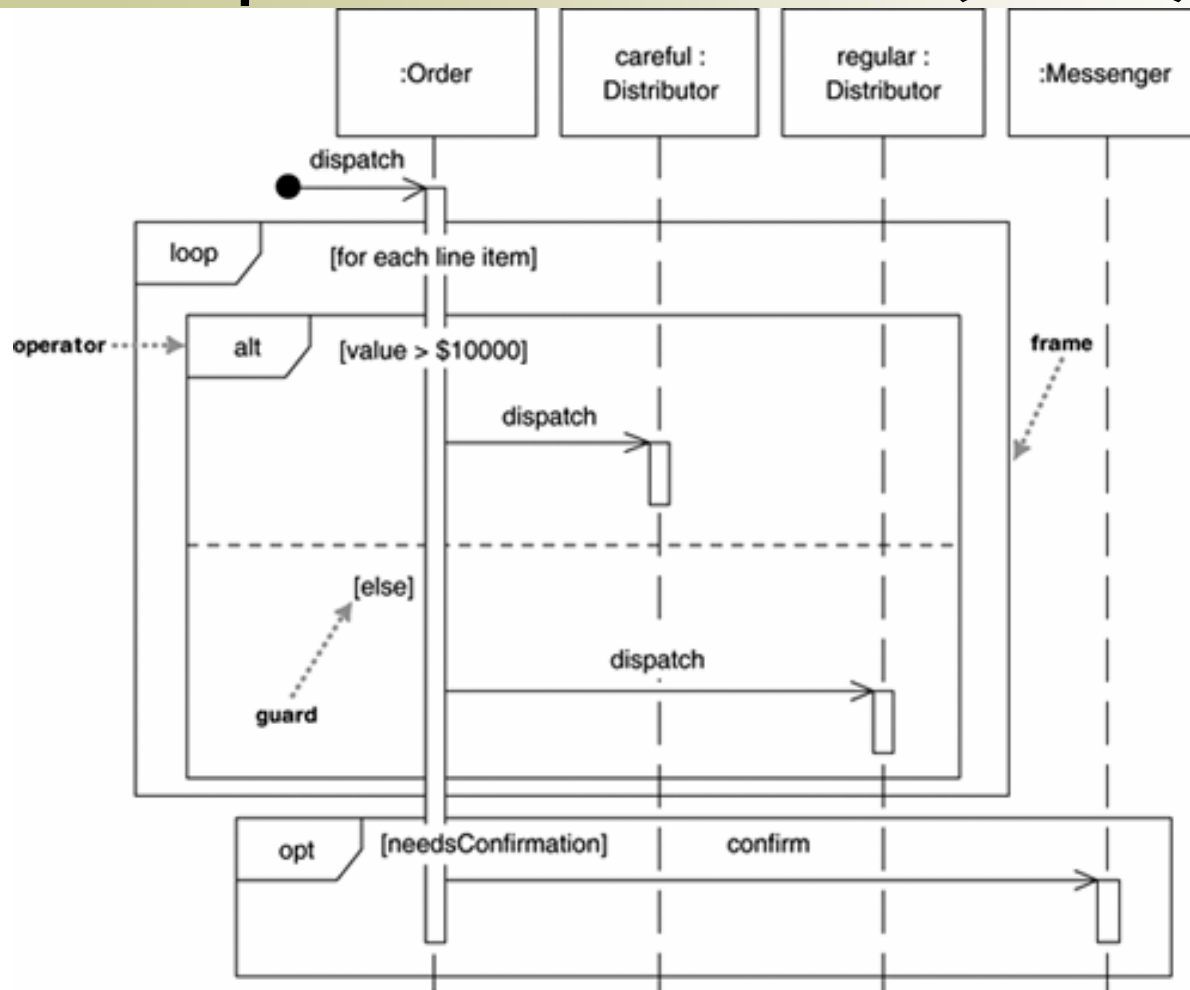
2.4 Creating and Deleting Participants



2.5 Loops, Conditionals

- A common issue with sequence diagrams is how to show looping and conditional behavior.
- The first thing to point out is that this isn't what sequence diagrams are good at. If you want to show control structures like this, you are better off with an *activity diagram* or indeed with code itself.
- That said, here's the notation to use. Both loops and conditionals use *interaction frames*, which are ways of marking off a piece of a sequence diagram.

2.5 Loops, Conditionals (cont')



2.5 Loops, Conditionals (cont')

- procedure dispatch
 - foreach (lineitem)
 - if (product.value > \$10K)
 - careful.dispatch
 - else
 - regular.dispatch
 - end if
 - end for
 - if (needsConfirmation) messenger.confirm
- end procedure

2.5 Loops, Conditionals (cont')

- **Common Operators** for Interaction Frames Operator
 - **Alt** : Alternative multiple fragments; only the one whose condition is true will execute
 - **Opt** : Optional; the fragment executes only if the supplied condition is true. Equivalent to an alt with only one trace
 - **Par** : Parallel; each fragment is run in parallel.
 - **Loop** : Loop; the fragment may execute multiple times, and the guard indicates the basis of iteration

2.5 Loops, Conditionals (cont')

■ Common Operators (cont')

- **Region** : Critical region; the fragment can have only one thread executing it at once.
- **Neg** : Negative; the fragment shows an invalid interaction.
- **Ref** : Reference; refers to an interaction defined on another diagram. The frame is drawn to cover the lifelines involved in the interaction. You can define parameters and a return value.
- **Sd** : Sequence diagram; used to surround an entire sequence diagram, if you wish.

2.5 Loops, Conditionals (cont')

- UML 1 used **iteration markers** and **guards**.
 - An iteration marker is a * added to the message name. You can add some text in square brackets to indicate the basis of the iteration.
 - Guards are a conditional expression placed in square brackets and indicate that the message is sent only if the guard is true.

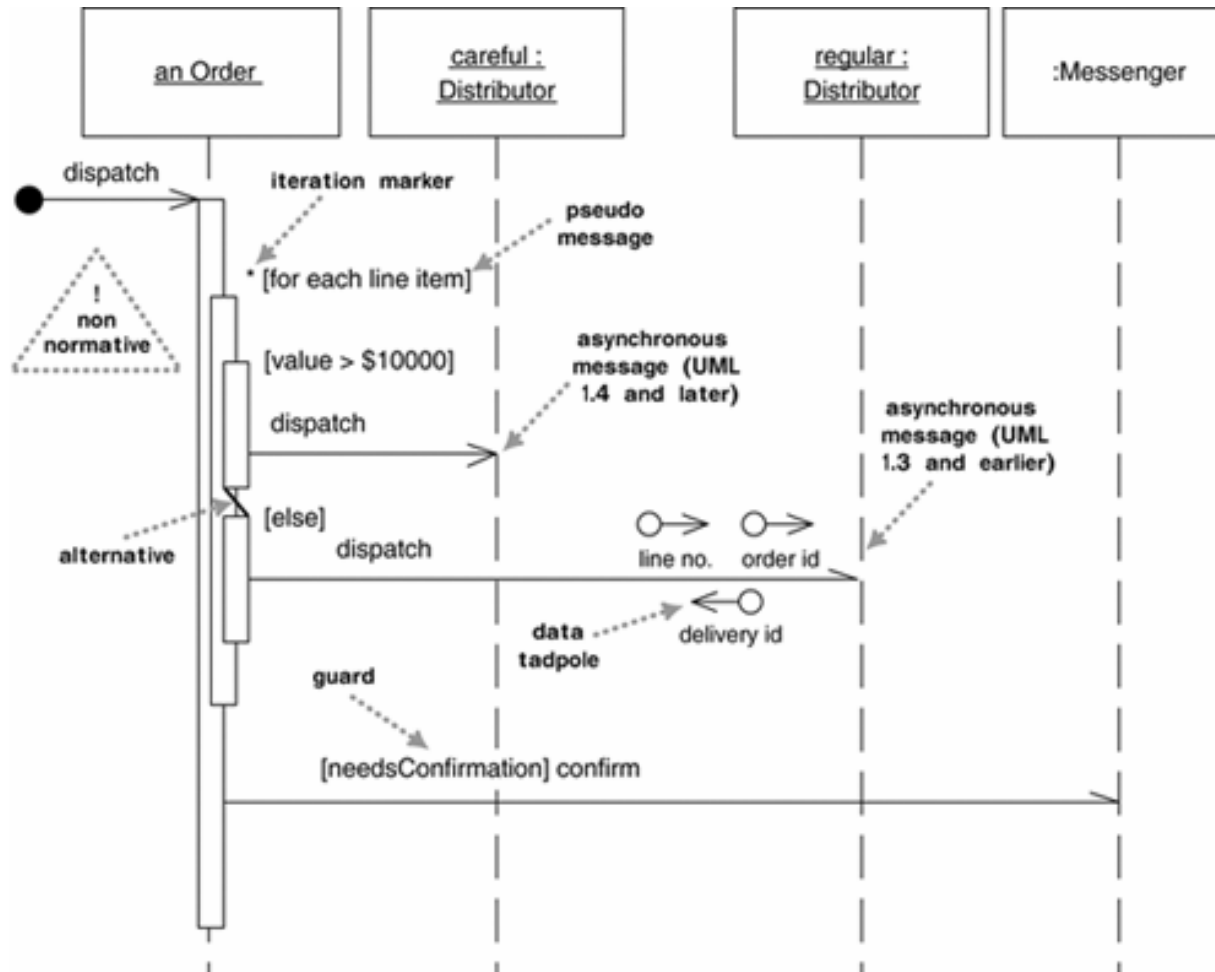
2.5 Loops, Conditionals (cont')

- *iteration markers* and *guards* have weaknesses
 - The guards can't indicate that a set of guards are mutually exclusive.
 - Both notations work only with a single message send and don't work well when several messages coming out of a single activation are within the same loop or conditional block.
- To get around this last problem, an unofficial convention that's become popular is to use a *pseudo message*, with the loop condition or the guard on a variation of the self-call notation.



2.5 Loops, Conditionals (cont')

- The UML standard has no graphic device to show passing data; instead, it's shown by parameters in the message name and return arrows.
- **Data tadpoles** have been around in many methods to indicate the movement of data, and many people still like to use them with the UML.

2.5 Loops, Conditionals (cont')



2.6 Synchronous and Asynchronous

- If a caller sends a *synchronous message*, it must wait until the message is done, such as invoking a subroutine. (filled arrowheads) 
- If a caller sends an *asynchronous message*, it can continue processing and doesn't have to wait for a response. (stick arrowheads) 
 - You see asynchronous calls in multithreaded applications and in message-oriented middleware. Asynchrony gives better responsiveness and reduces the temporal coupling but is harder to debug.

2.7 When to Use Sequence Diagrams

- Use sequence diagrams when you want to look at the behavior of several objects within a single use case.
- Sequence diagrams are good at showing collaborations among the objects; they are not so good at precise definition of the behavior.

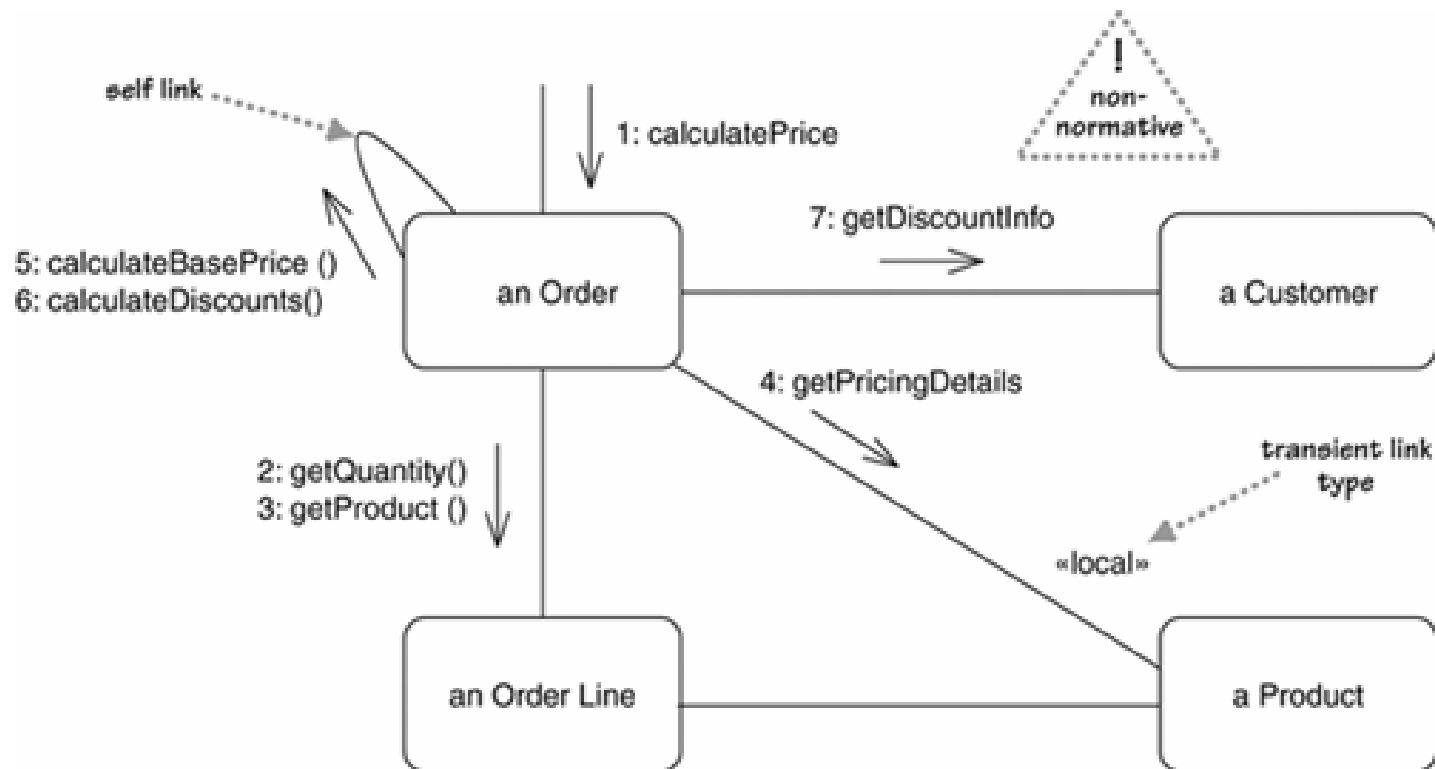
2.7 When to Use Sequence Diagrams

- If you want to look at the behavior of a single object across many use cases, use a **state diagram**.
- If you want to look at behavior across many use cases or many threads, consider an **activity diagram**.
- If you want to explore multiple alternative interactions quickly, you may be better off with **CRC (Class-Responsibility-Collaboration) cards**, as that avoids a lot of drawing and erasing.

[3. Communication Diagrams]

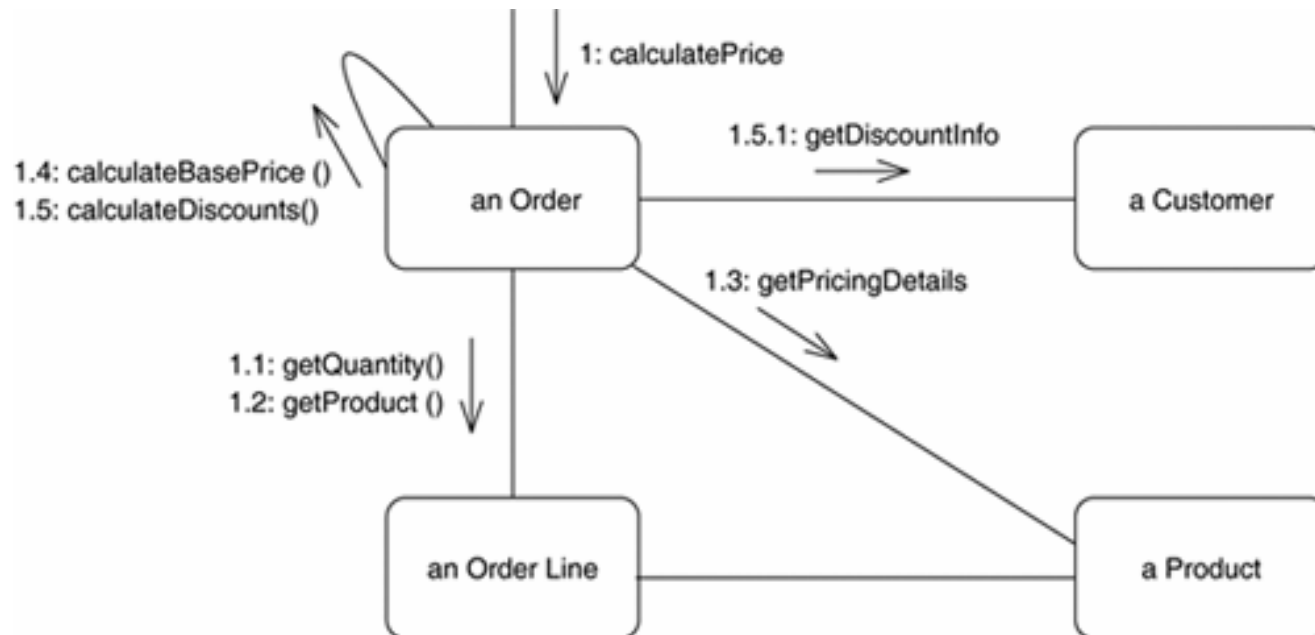
- *Communication diagrams*, a kind of interaction diagram, emphasize the data links between the various participants in the interaction.
- communication diagram allows free placement of participants, allows you to draw links to show how the participants connect, and use numbering to show the sequence of messages.
- In UML 1.x, these diagrams were called collaboration diagrams.

3. Communication Diagrams (cont')



3. Communication Diagrams (cont')

- Legal UML : you have to use a nested decimal numbering scheme



[3. Communication Diagrams (cont')]

- A strong part of the decision is personal preference: Some people like one over the other.
- A more rational approach says that sequence diagrams are better when you want to emphasize the sequence of calls and that communication diagrams are better when you want to emphasize the links.

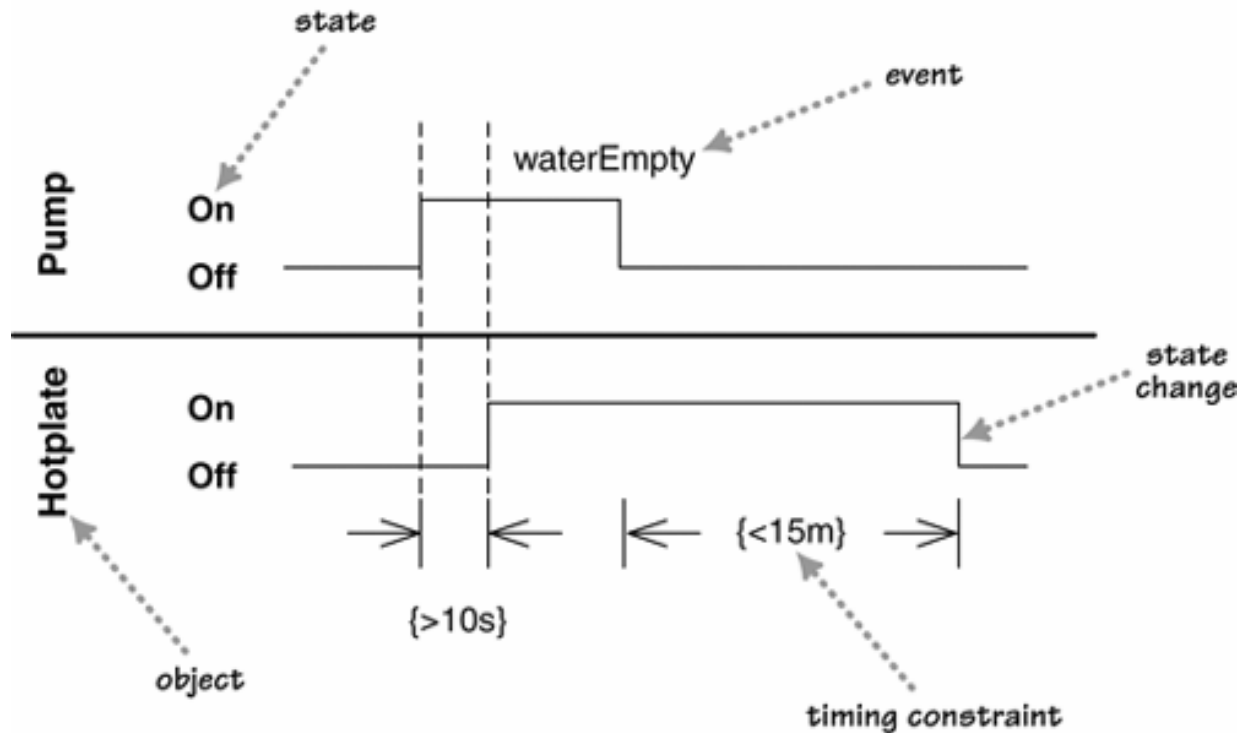
4. Timing Diagrams

- *Timing diagrams* are another form of interaction diagram, where the focus is on timing constraints: either for a single object or, more usefully, for a bunch of objects.

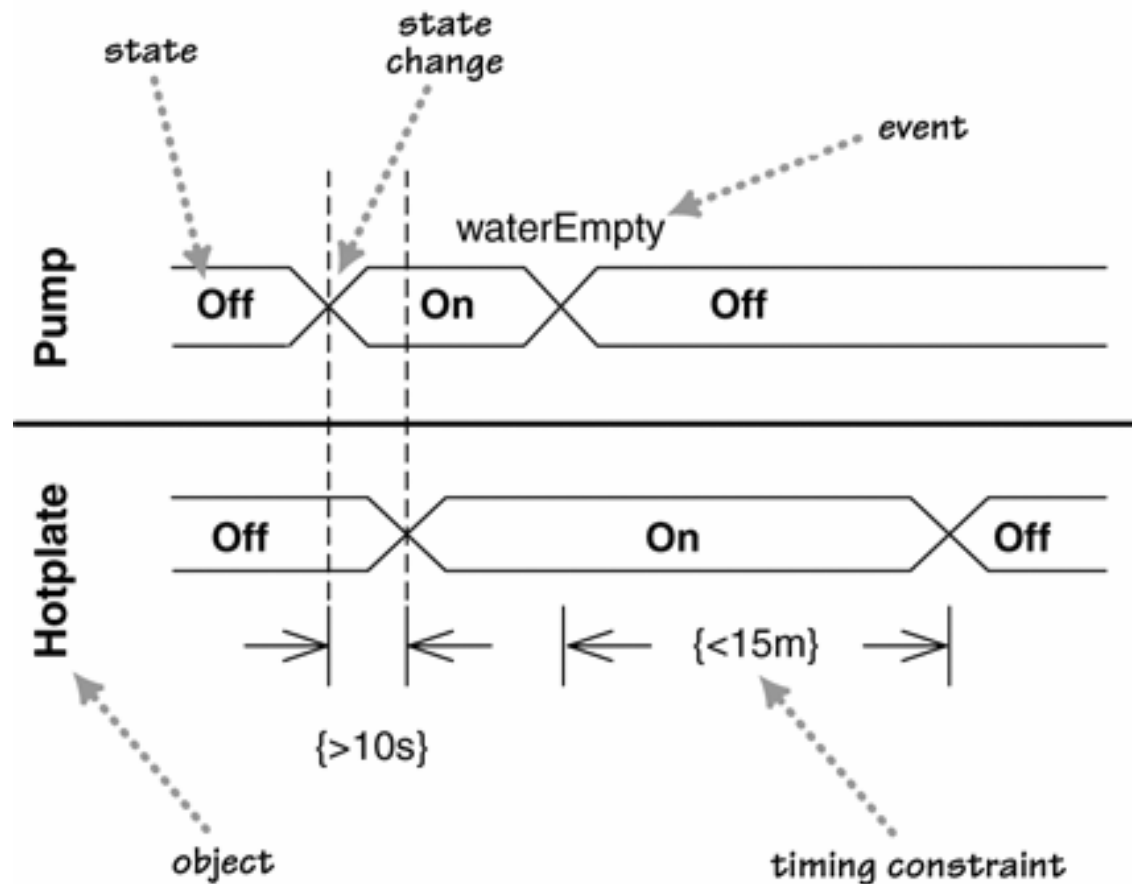
4. Timing Diagrams (cont')

- Let's take a simple scenario based on the pump and hotplate for a coffee pot.
- Let's imagine a rule that says that at least 10 seconds must pass between the pump coming on and the hotplate coming on. When the water reservoir becomes empty, the pump switches off, and the hotplate cannot stay on for more than 15 minutes more.

4. Timing Diagrams (cont')



4. Timing Diagrams (cont')

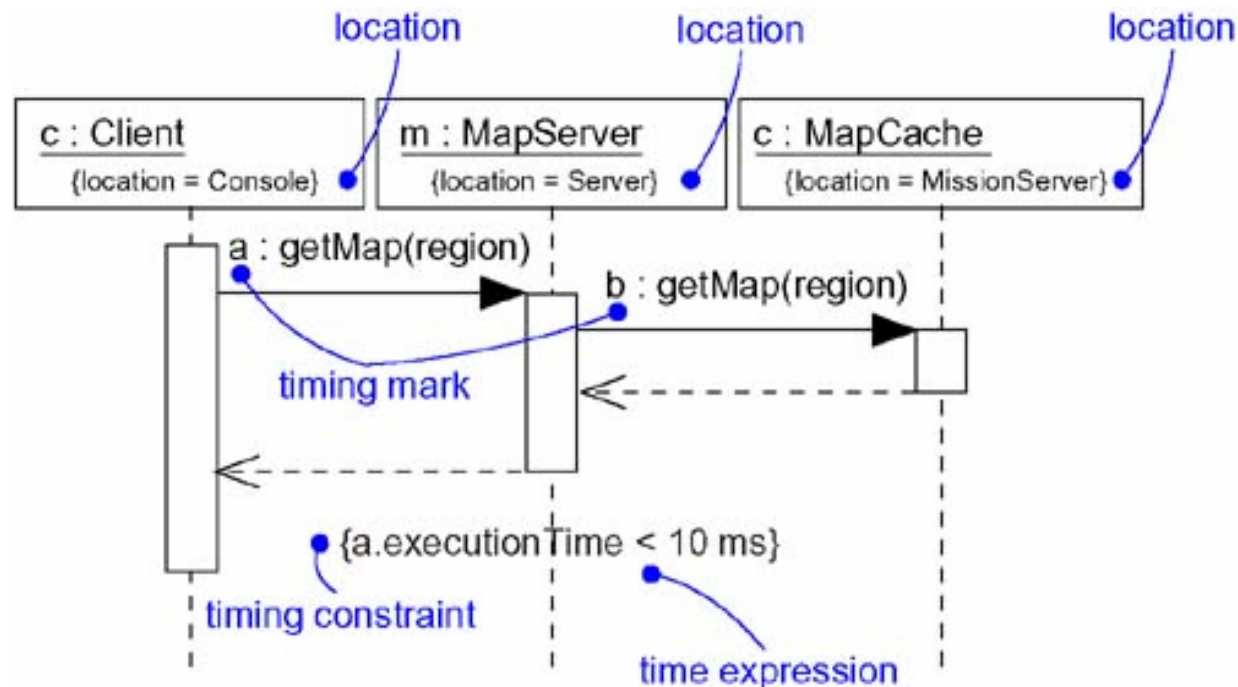


4. Timing Diagrams (cont')

- When to Use Timing Diagrams
- Timing diagrams are useful for showing timing constraints between state changes on different objects. The diagrams are particularly familiar to hardware engineers.

5. Time & Space

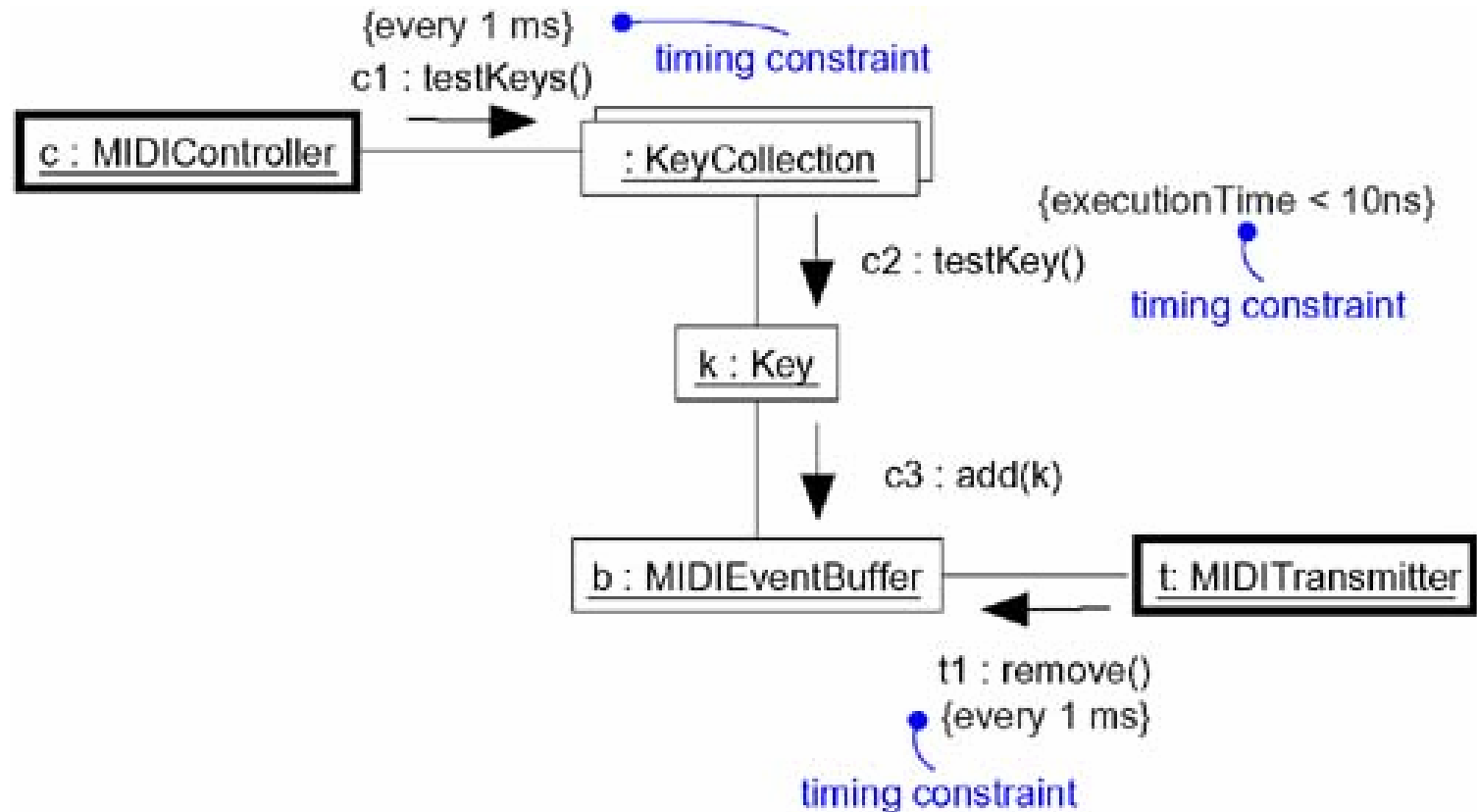
- To represent the modeling needs of real time and distributed systems, the UML provides a graphic representation for timing marks, time expressions, timing constraints, and location.



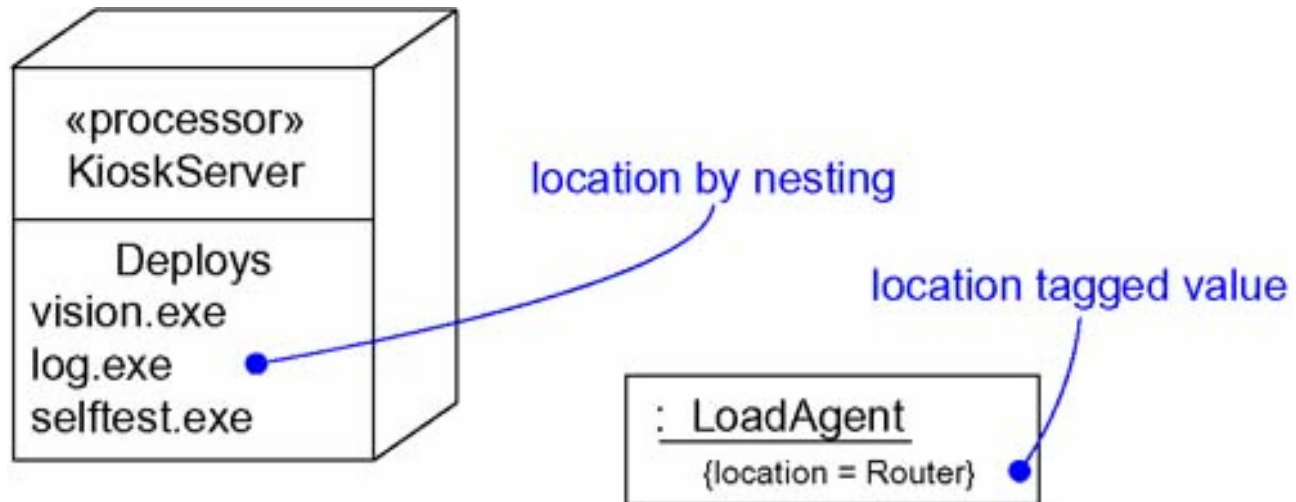
5.1 Time

- A *timing mark* is a denotation for the time at which an event occurs.
 - Graphically, a timing mark is formed as an expression from the name given to the message (which is typically different from the name of the action dispatched by the message).
- A *time expression* is an expression that evaluates to an absolute or relative value of time.
- A *timing constraint* is a semantic statement about the relative or absolute value of time.
 - Graphically, a timing constraint is rendered as for any constraint—that is, a string enclosed by brackets and generally connected to an element by a dependency relationship.

5.1 Time (cont')



5.2 Location

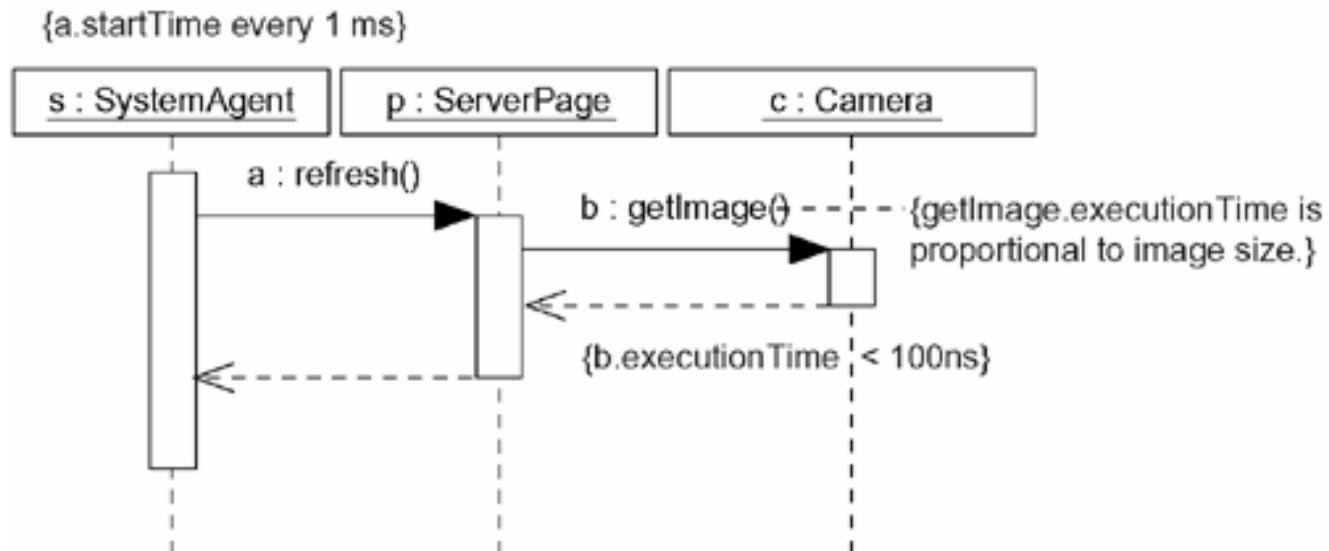


5.3 Modeling Timing Constraints

- For each event in an interaction, consider whether it must start at some absolute time.
- For each interesting sequence of messages in an interaction, consider whether there is an associated maximum relative time for that sequence.

5.3 Modeling Timing Constraints

- three functions of the message
 - startTime
 - stopTime
 - and executionTime.



Assignment 2

- 为“位图浏览工具”的两项功能“位图放大（Zoom In Bmp）”和“移动位图文件（Move Bmp）”设计顺序图和交互图。
 - Zoom In 的功能描述如下： 如果用户将鼠标移动至位图内部，然后按下鼠标左键移动，则在显示窗口内用橡皮带的方式画出一个矩形，当释放鼠标左键后，将此矩形内部的内容放大显示至显示窗口内部。
 - Move的功能描述如下： 用户在位图区域内按下鼠标左键；保持左键按下拖动鼠标；释放鼠标左键。
- 注意：
 - 两个用例都具有2个Actor，用户和显示窗口
 - 参与交互的对象实现接受、处理、数据的分离
 - 注意基本的条件判断（如鼠标是否在位图之外）