Chapter 4 Sequence Diagrams

- interaction diagrams describe how some objects interact or collaborate, possibly in the execution of a use case scenario
- participants usually but not always objects since UML 2, also names not underlined anymore
- two varieties: sequence diagrams and communication diagrams
- most common is sequence diagrams
- with sequence diagram the time ordering of message sending between objects is obvious
- name and class optional

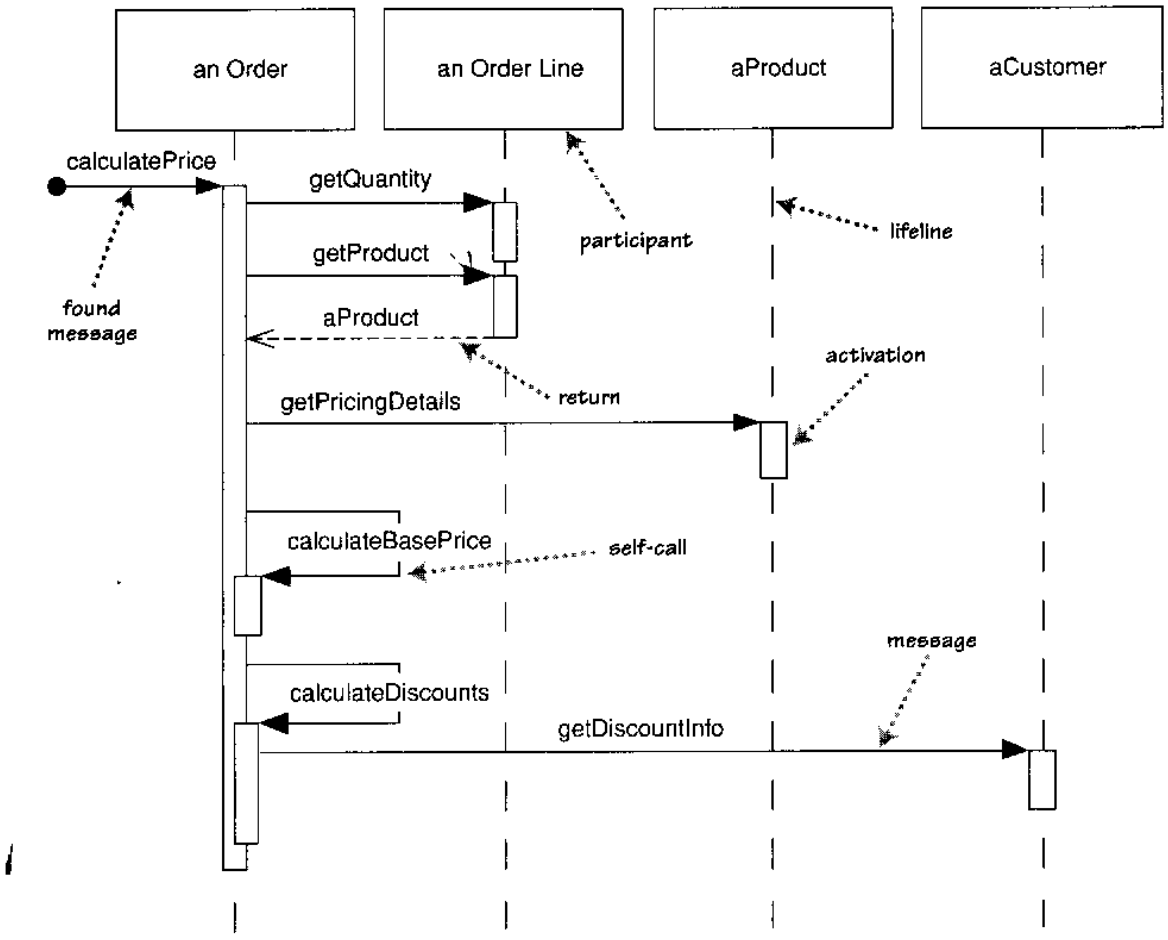

Figure 4.1 *A sequence diagram for centralized control*

- activation bar shows that object is active, i.e. that one of its methods is on the stack in an intermediate state of execution, also optional
- use return arrows only when information is added
- found message – undetermined source

Alternative Diagram

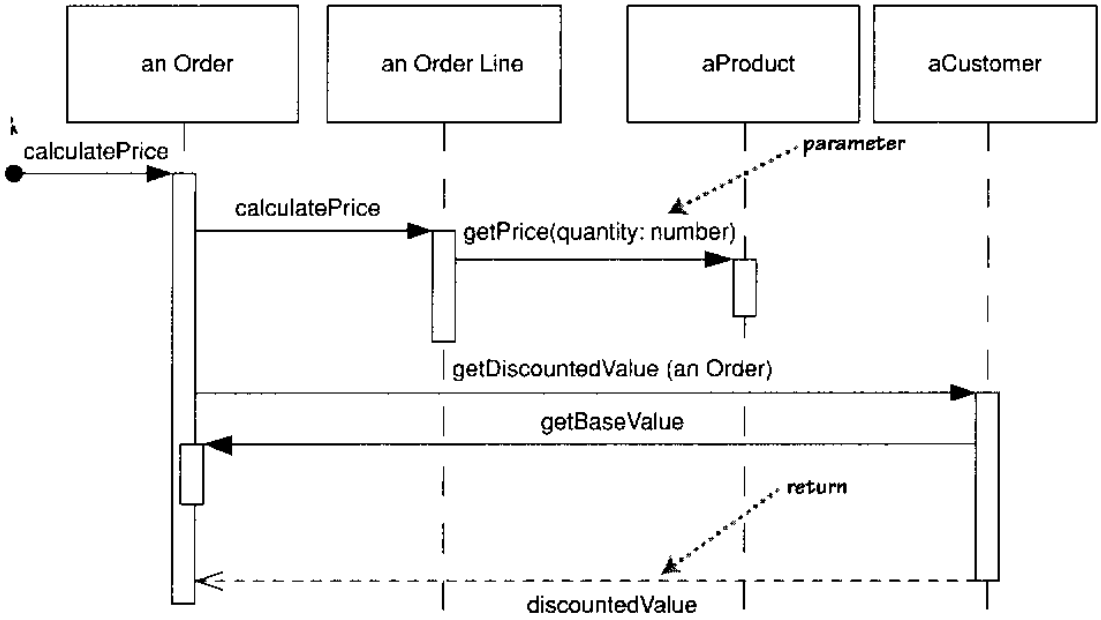

Figure 4.2 *A sequence diagram for distributed control*

- consider fig 4.2, has same functionality as fig 4.1
- these diagram show clearly differences in object interaction
- weak point is not good at showing algorithm detail
- notice centralised versus distributive control
- centralised preferred traditionally, processing controlled from one place, gives more of a sense of a program

Distributed Control

- more favoured by OO community, helps localise effects of change as data and behaviour that access it often change together
- also it is more conducive to use of polymorphism and avoiding conditional logic, e.g. getPrice() can be more easily overridden in a *Product* subclass and used in conjunction with polymorphism rather than a writing a complicated case statement in the *Order* class.
- in general, OO paradigm favours smaller objects with many small methods rather than big procedures – an OO design principle if you like

Creating and Deleting Participants

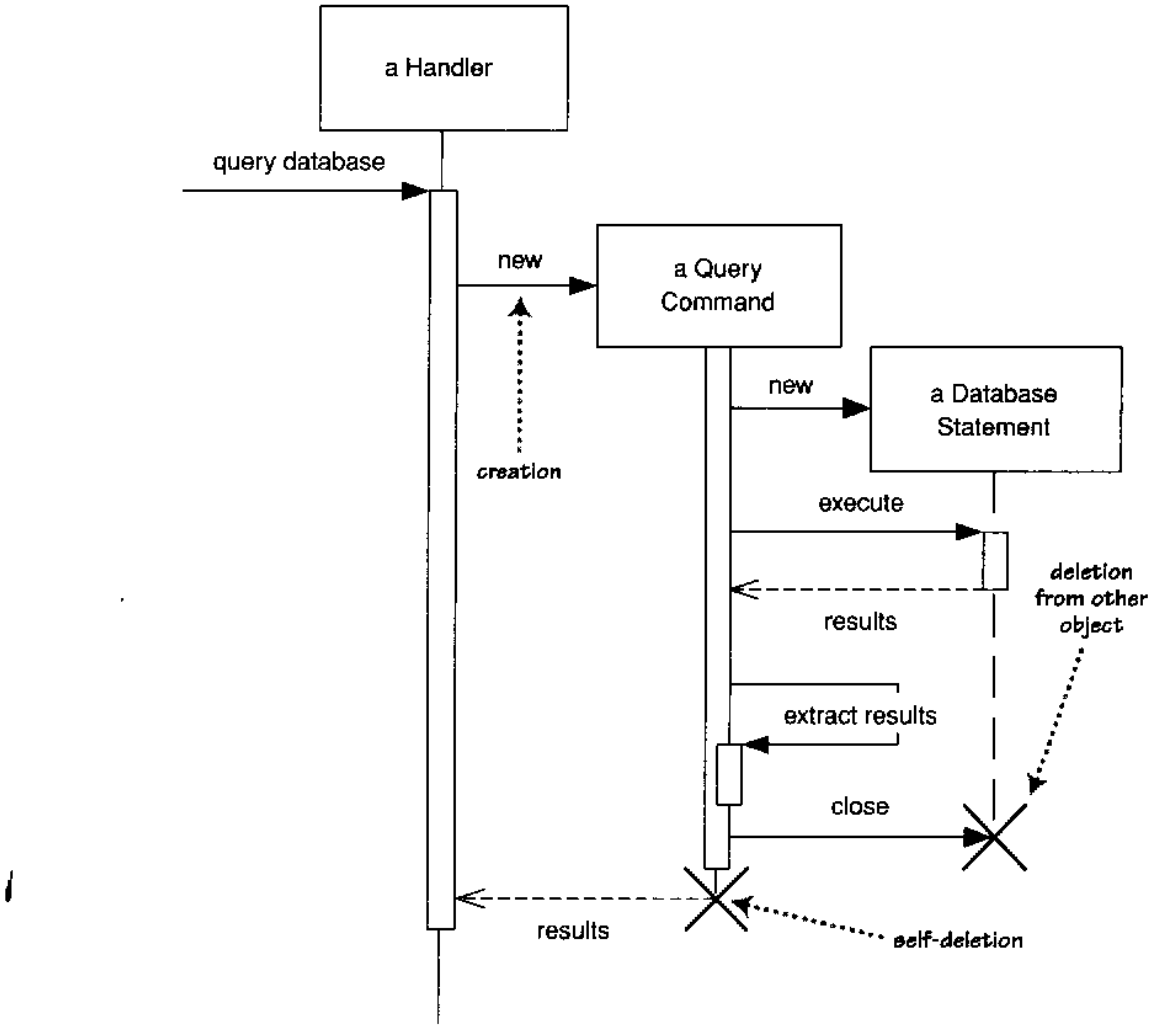

Figure 4.3 *Creation and deletion of participants*

- **X** denotes that one participant explicitly deletes another
- not done by programmer in garbage-collected environments such as Java (C++ is different)

Loops and Conditionals

- not good at showing control logic – better use activity diagram or pseudocode or indeed code

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

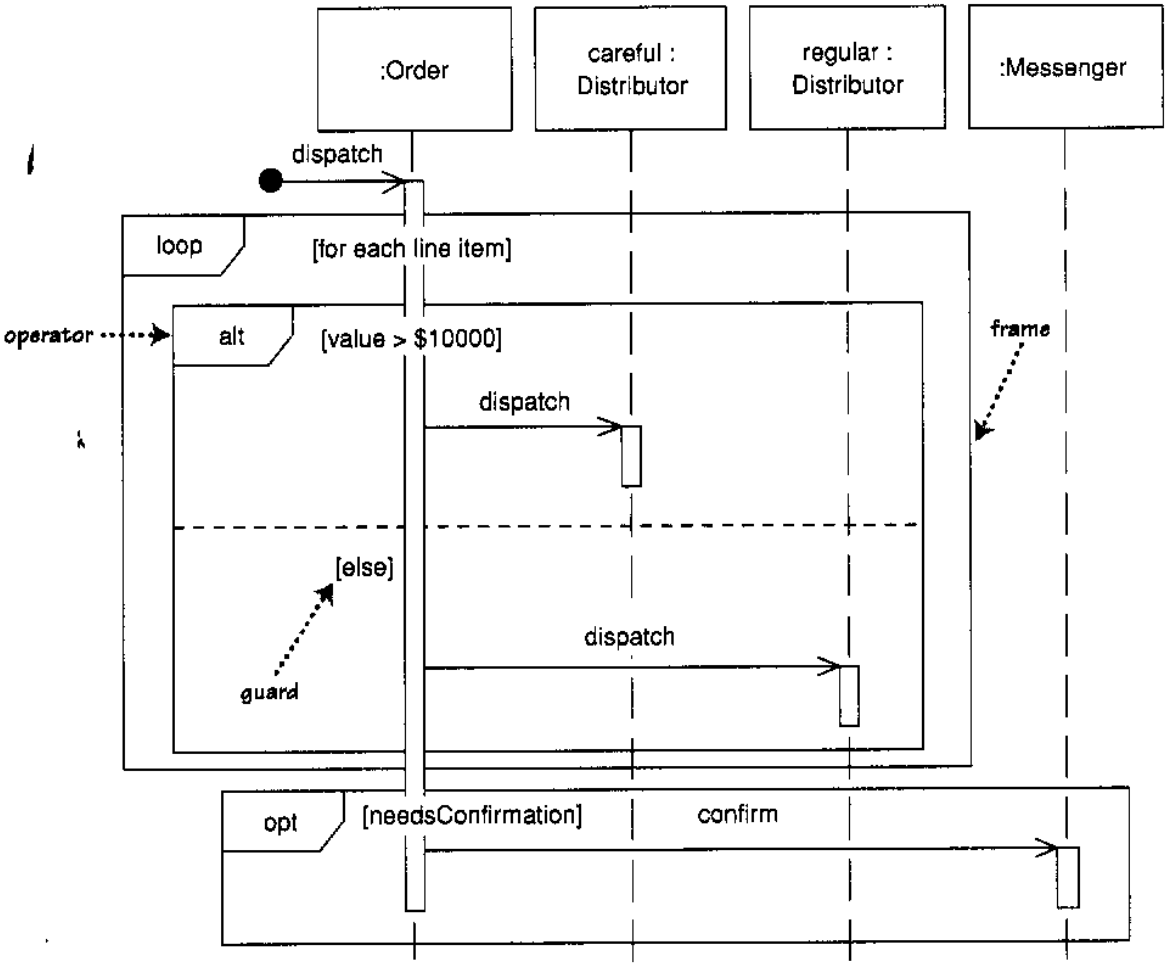

Figure 4.4 *Interaction frames*

- sequence diagram has frames (new in UML 2), frame divided into fragments, each of which may have a guard. Frame has an operator
- UML 1 uses iteration markers and guards. * added to message. Dropped in UML 2

Table 4.1 *Common Operators for Interaction Frames*

Operator	Meaning
alt	Alternative multiple fragments; only the one whose condition is true will execute (Figure 4.4).
opt	Optional; the fragment executes only if the supplied condition is true. Equivalent to an alt with only one trace (Figure 4.4).
par	Parallel; each fragment is run in parallel.
loop	Loop; the fragment may execute multiple times, and the guard indicates the basis of iteration (Figure 4.4).
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.

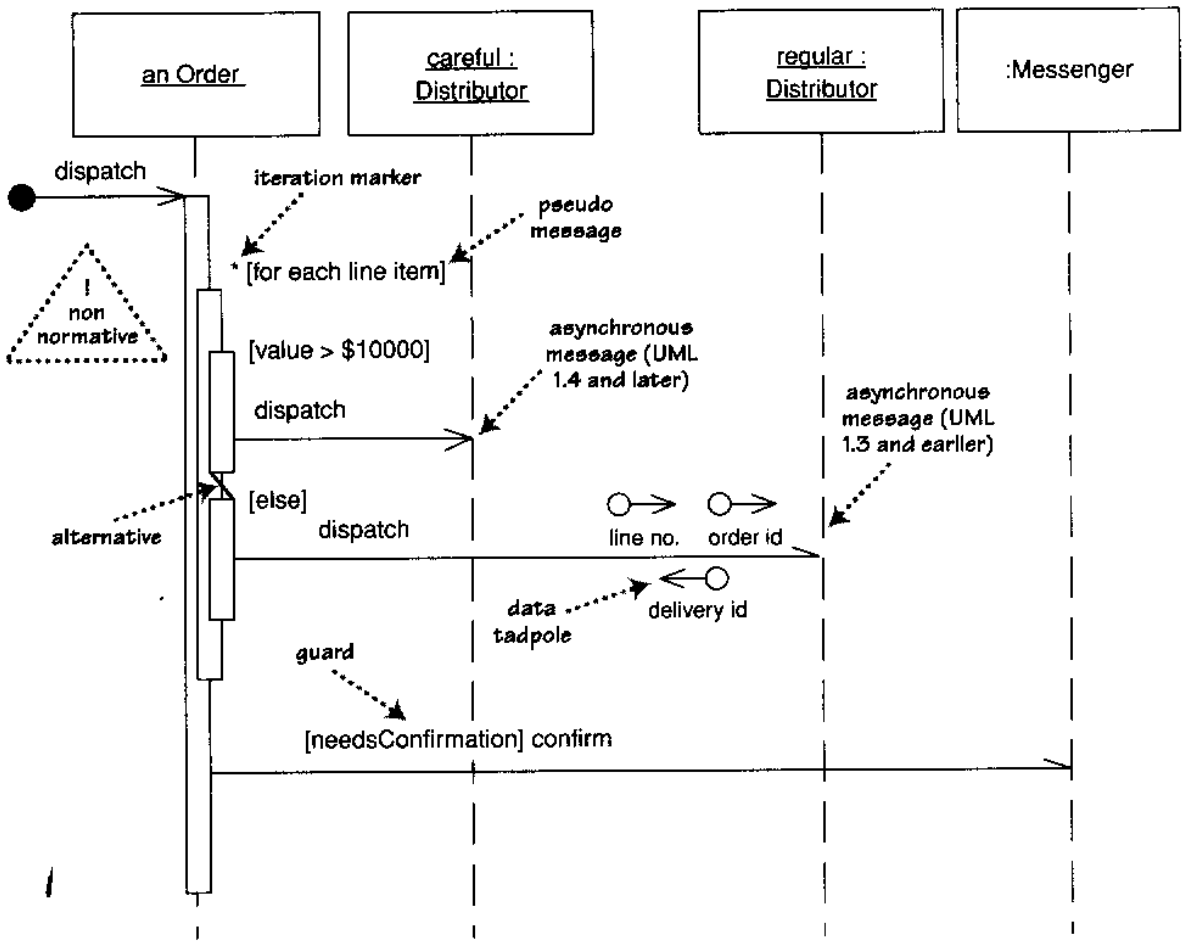

Figure 4.5 *Older conventions for control logic*

- use of pseudomessage useful – fig 4.5
- may drop activation bar for simple method calls
- Fowler finds adding control info to sequence diagrams more more useful than pseudocode

Synchronous and Asynchronous Calls

- synchronous – calling method must wait for target method to complete
- asynchronous – caller can continue processing, does not have to wait. Used in multithreaded applics and message oriented middleware.
- asynchrony gives better responsiveness but more difficult to debug
- Fowler recommends using obsolete (in UML 2) half-stick, more noticeable

When to use

- good at showing collaboration, not for precise behaviours, useful for a single use case
- to see single object behaviour across use cases, statechart diagrams are useful
- for behaviours of more than one object across many threads or many use cases, an activity diagram is useful.

Class, Responsibility, Collaboration (CRC) Cards

- for quick exploration of multiple alternative interactions, CRC cards may be useful. Useful in exploring design alternatives, later on use sequence diagrams to document the agreed interaction
- drawing sequence diagrams too slow for exploration, use of CRC cards more animated
- explore behaviour rather than data
- focus on responsibilities moves one away from notion of class as dumb data holder, helps with understanding higher level behaviour of class
- responsibility refers to an undetermined clump of attributes and operations
- common mistake is long list of low level responsibilities, hence size of card

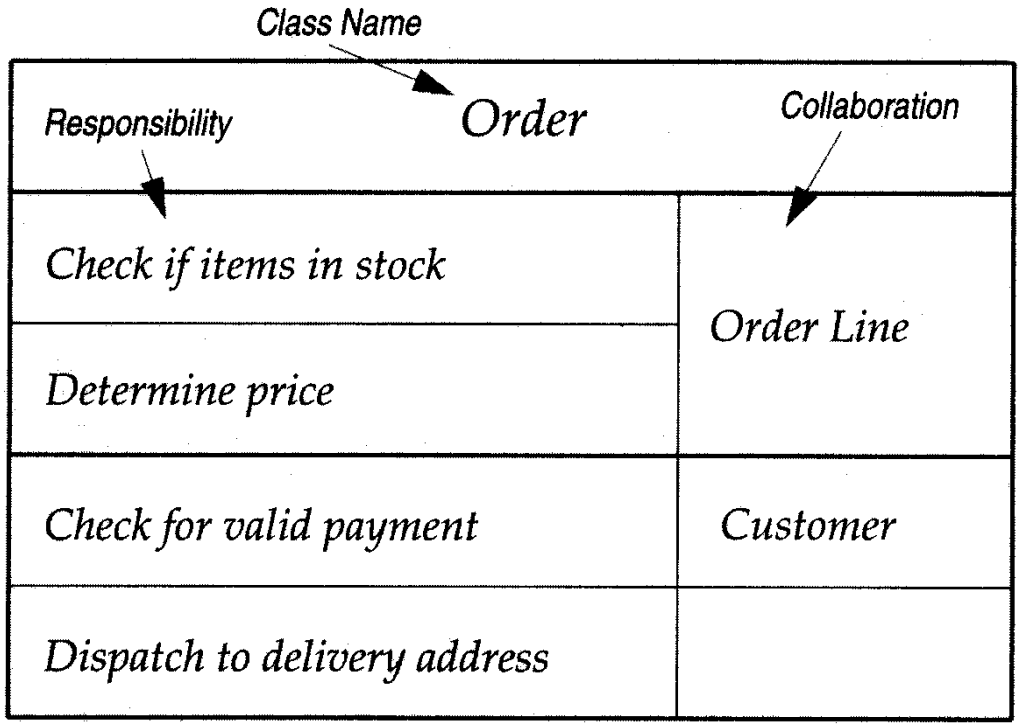

Fig 4.6 – CRC card