Figure 13.1 The interaction of use cases through objects

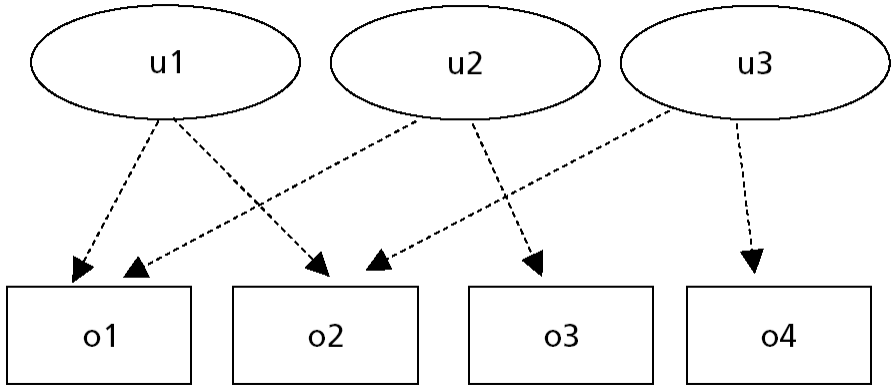

Figure 13.2 The primary inputs and outputs of design

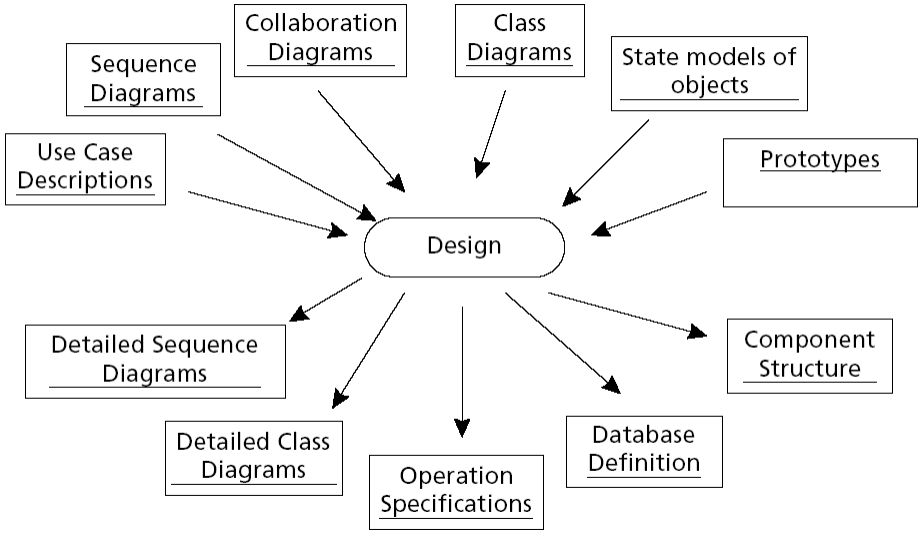

Figure 13.6 An object lifeline

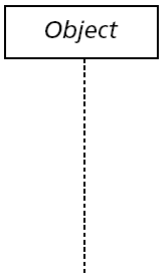

Figure 13.7 Locus of control bar on a lifeline

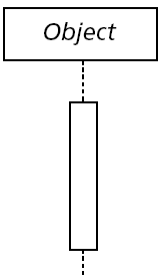

Figure 13.8 Initiating locus of control

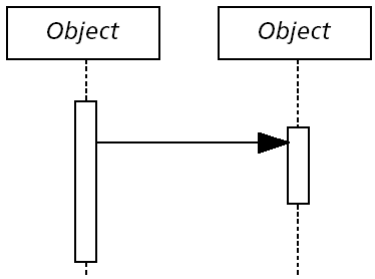

Figure 13.9 Showing an object creating another object

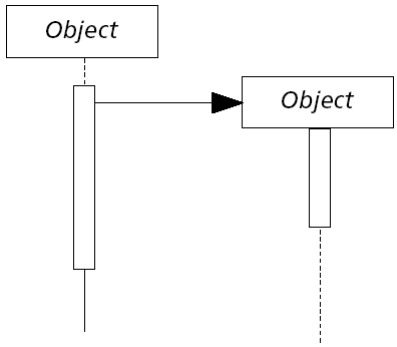

Figure 13.10 One object destroying another

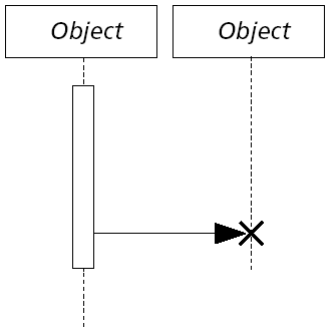

Figure 13.11 An object calling itself

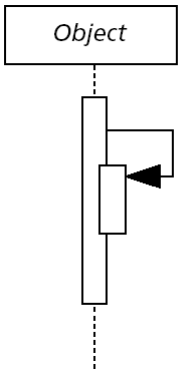

Figure 13.12 Iteration on a sequence diagram

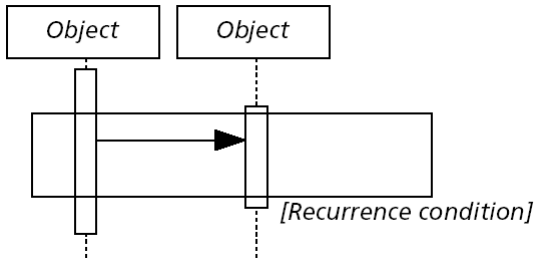

Table 13.6 Message flow notation for sequence diagrams

Arrow	Stimulus	Description
→	Procedural or Synchronous	When the first object calls the second object, it suspends activity until the second object completes the request. When the second object loses the locus of control, the first one resumes.
→	Flat	This triggers the second object to start execution. Normally this is asynchronous, meaning that the first object does not wait for a return. However, it can be used if it is unknown whether or not the message is asynchronous.
↘	Asynchronous	This triggers execution of the second object, and the first object continues without waiting for the second object to complete its task.
[c1] → / [c2] →	Branching	Splitting the arrow into two or more branches means that more than one object might be called, depending on some conditions. The conditions are written in square brackets. If the conditions are mutually exclusive, this represents a branch; otherwise it shows potential concurrent execution of the called objects.

Figure 13.13 Standard design pattern for interactive systems

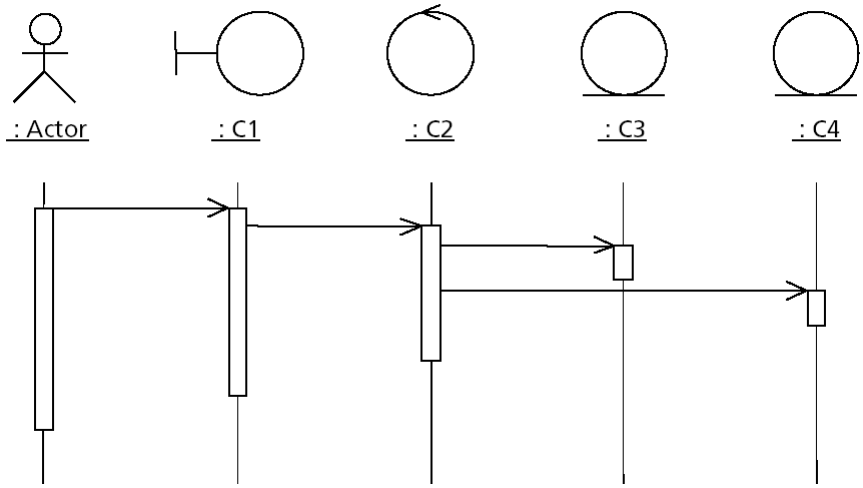

Figure 13.14 A simpler pattern for simple interactive systems

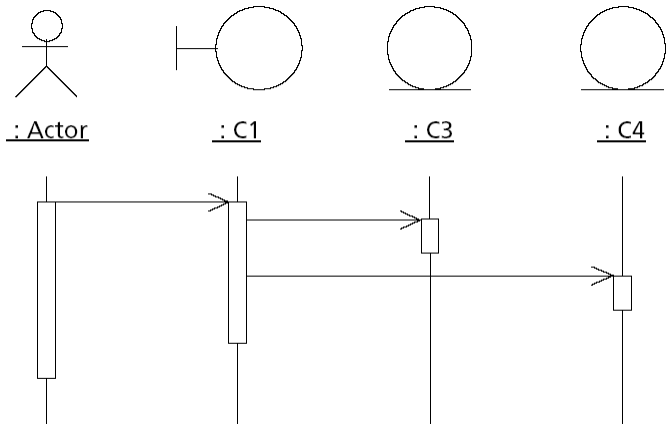

Figure 13.15 A pattern that shows the use of factories to instantiate objects that have been stored in a database

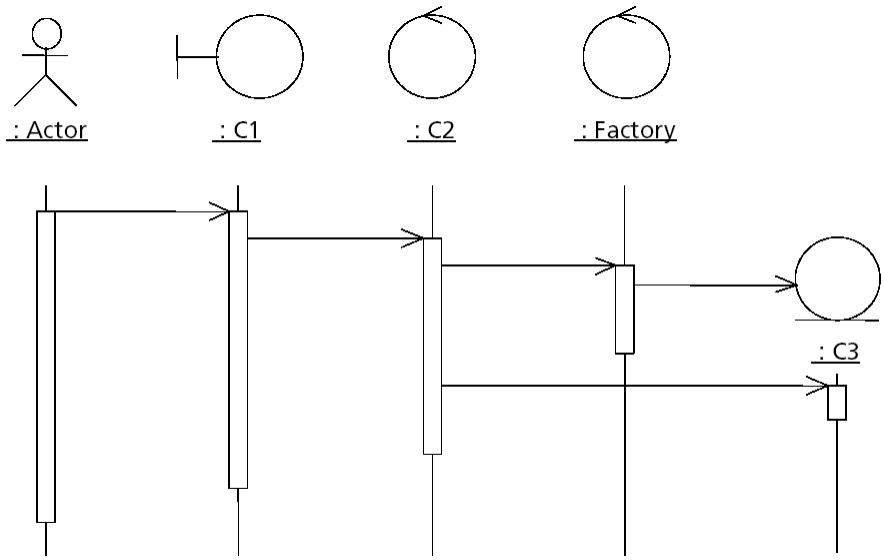

Figure 13.16 A collaboration diagram

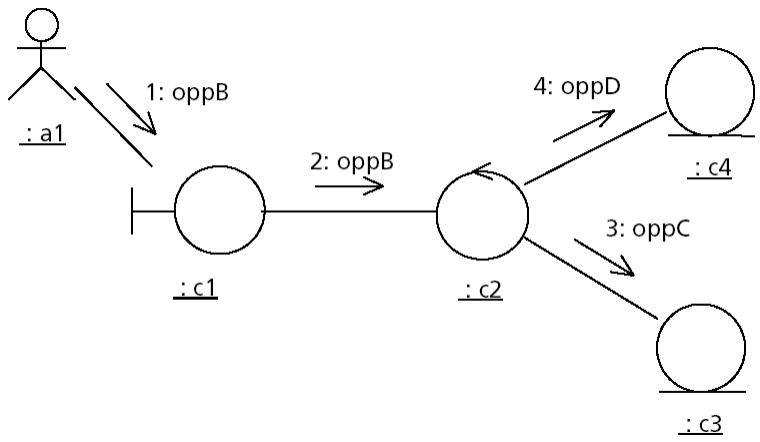

Figure 13.17 Pseudo-code to describe invoice printing

```
Print Invoice
SalesLedger.open
SalesLedger.getSalesItem
While SalesLedger.hasItems
  Customers.getCustomer(salesItem.customerID)
  Print the customer details on the invoice
  While salesItem.customer unchanged
    Print the current salesItem
    Add salesItem.value to total
    SalesLedger.getSalesItem
  EndWhile
  Print invoice salesItems total
  Apply customer discounts
  Calculate VAT
  Print amount payable
  Skip to next invoice
EndWhile
```


Figure 13.18 An activity diagram to specify operation logic

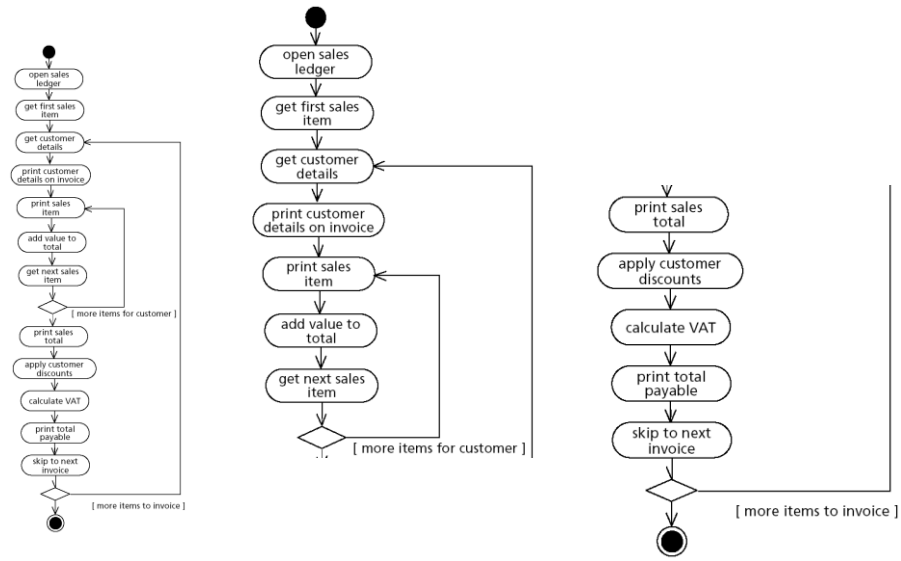

Figure 13.21 UML notation for a component

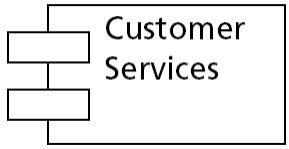

Figure 13.22 UML notation for an interface class

Figure 13.23 An interface class linked to a component, publishing the services of the component

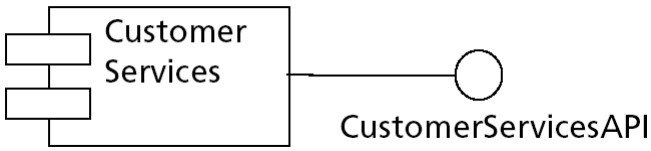

Figure 13.24 The organization of components to define the overall structure of the application

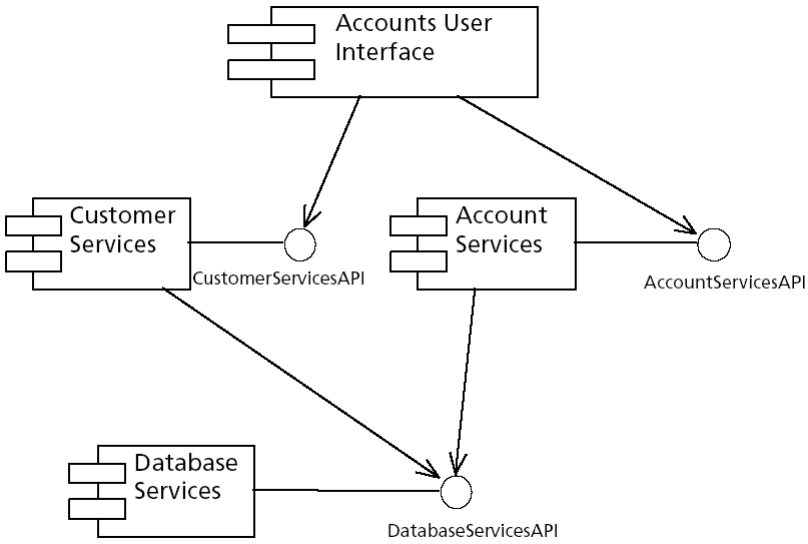

Figure 13.25 The workflow for component-based development

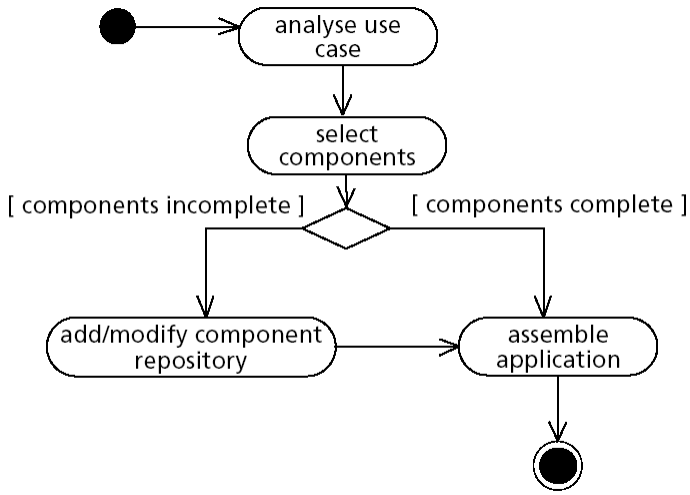

Figure 13.26 Use of a framework

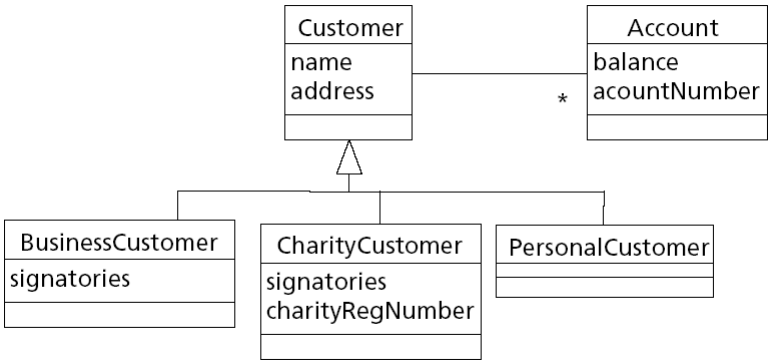

Figure 13.27 A simple pattern to show the typical structure of an order

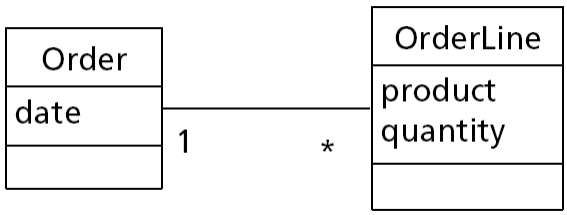

Figure 13.28 Layout of the welcome screen for the Validate Customer use case

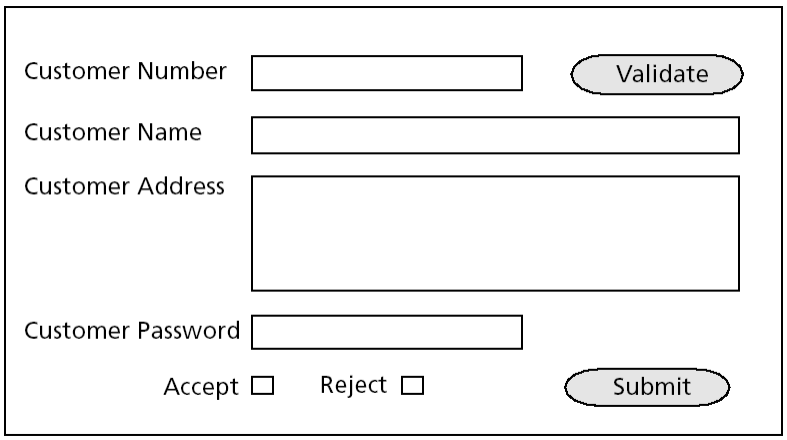

Figure 13.29 A UML representation of the welcome screen

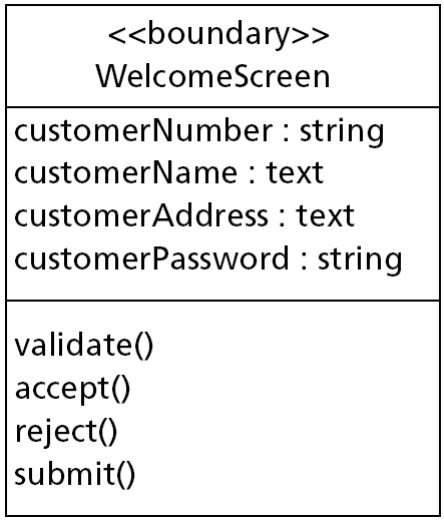

Figure 13.30 The addition of a factory object to create an object from the database

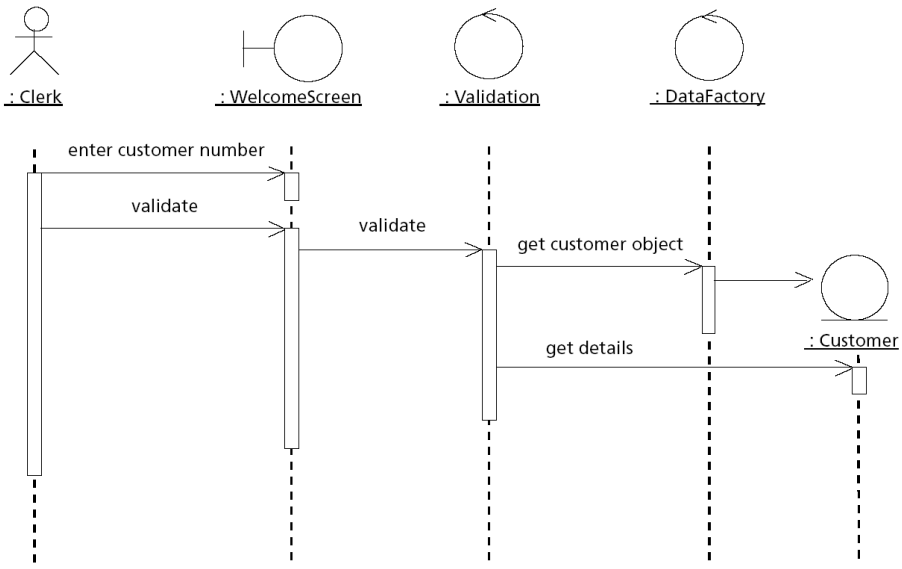

Figure 13.31 Class definition for the Validation control class

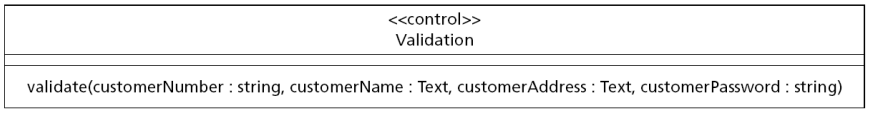

Figure 13.32 The DataFactory class definition.

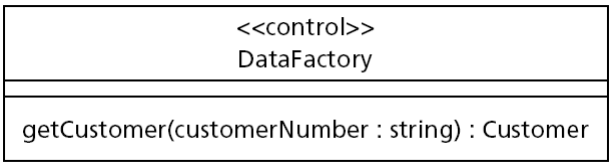

Figure 13.33 Class definition for a customer object