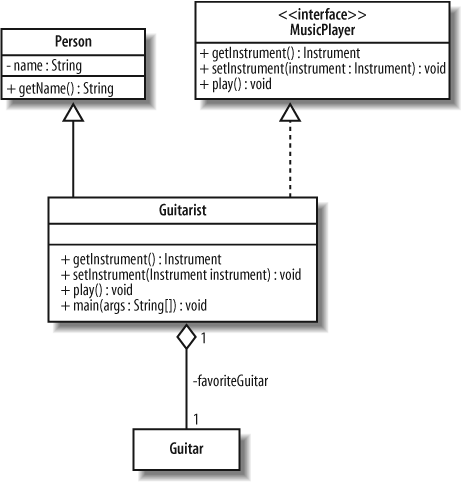
# Learning UML 2.0

## Introduction

### Example



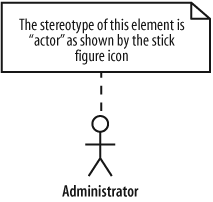
### Degree of UML

As a sketch

As a blueprint

As a programming language

Stereotypes signify a special use or intent and can be applied to almost any element of UML notation.

Note the administrator is an actor and a person

Standard stereotypes include

Utility

### Service

Subsystem

Executable

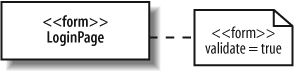
File

Library

Source

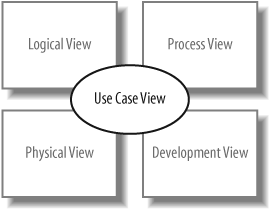
### Tagged values

Stereotypes can contain extra information that relates to the element to which they are applied



## Modelling requirements: use cases

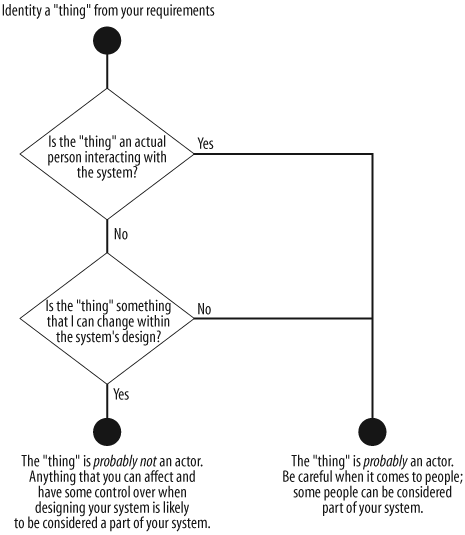
Use cases are at heart of your model



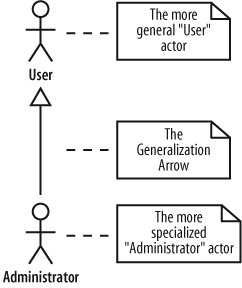
Actors

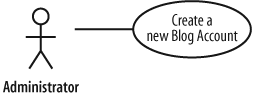


Questions to ask for identifying an actor

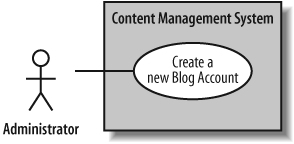


Use cases and communication lines





System boundaries



Information can be included in use case description

Description detail

Related requirements

Goal in context

Preconditions

Successful end condition

Failed end condition

Primary actors

Secondary actors

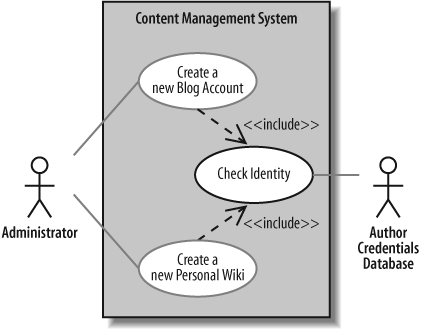
Trigger

Main flow

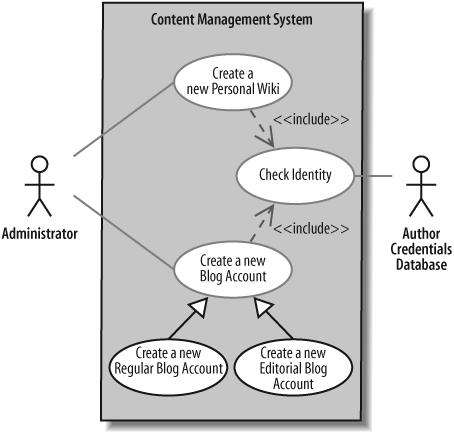
Extensions

### Use case relationships

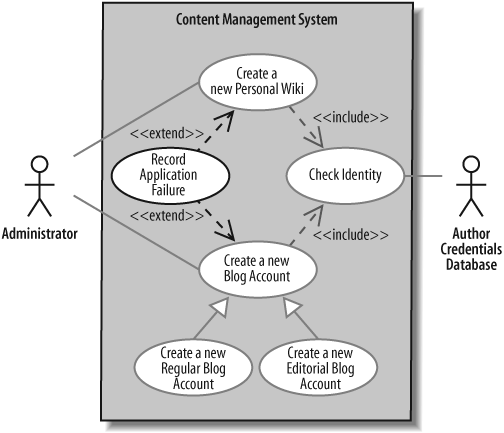
The <<include>> relationship



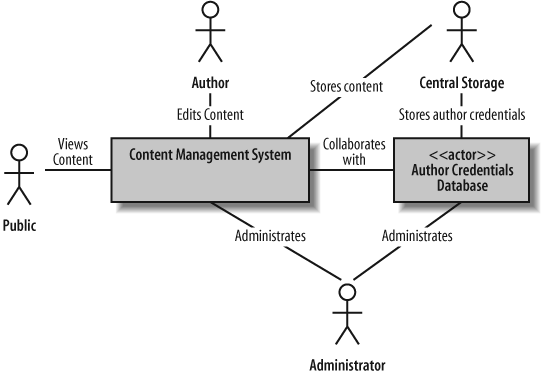
Special cases, example create regular blog account



The <<extend>> relationship, a means for you to show that a use case might completely reuse another use case’s behavior

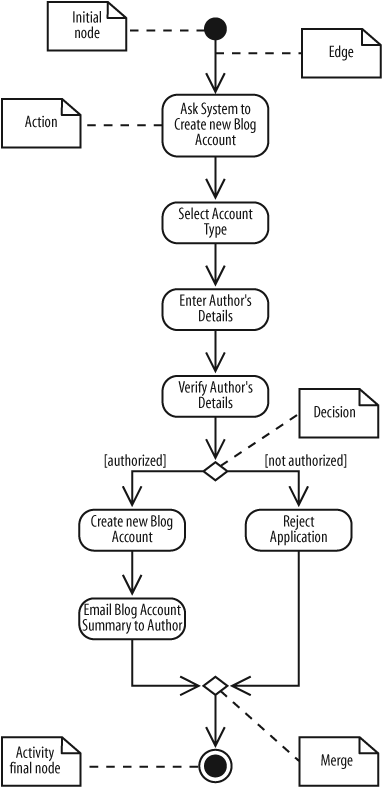


Use case overview diagrams

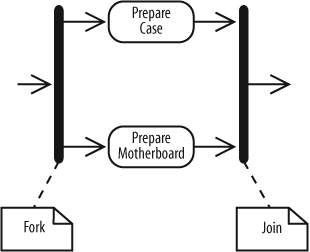


## Activity diagram

Example, flow the use case flow



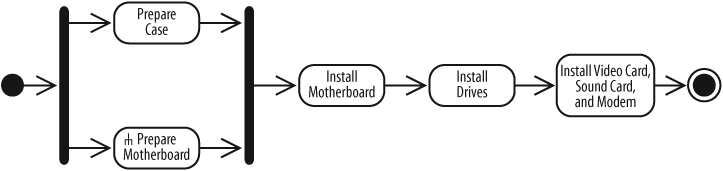
Multiple tasks at same time

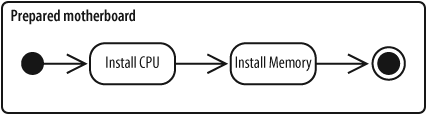


Time events

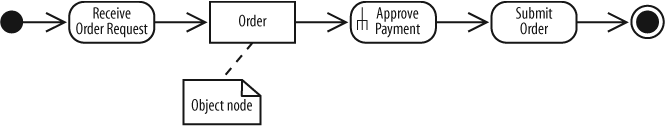


Call another activity

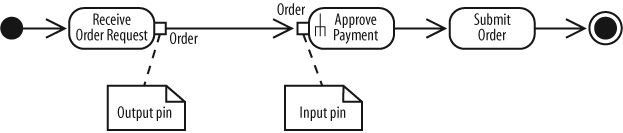




Showing objects passed between actions



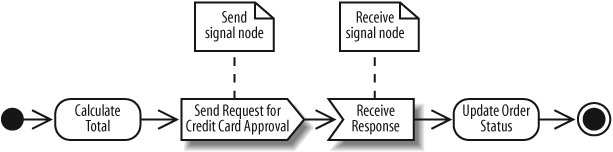
Showing action inputs and outputs



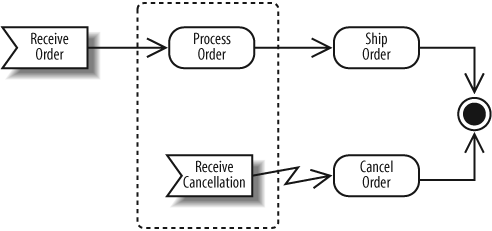
Showing how objects change state during an activity

The focus of this diagram is the change of state of the Order object throughout the order approval process

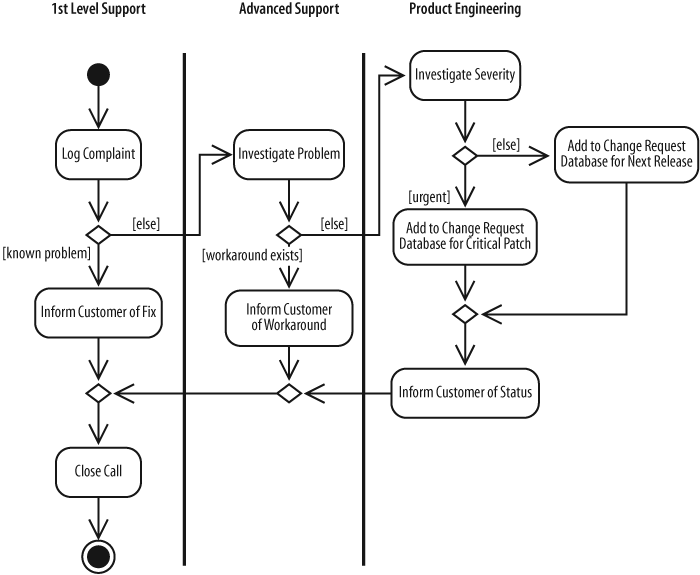
Sending and receiving signals



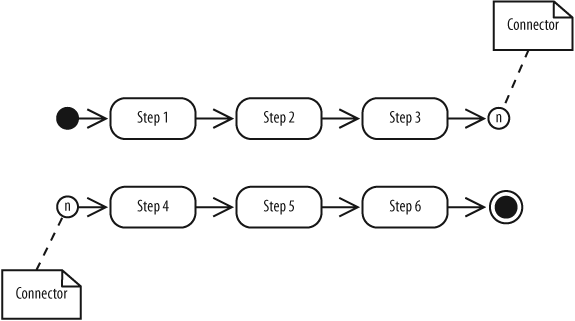
Interrupting an activity, a process that can be interrupted



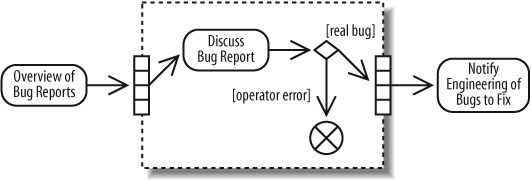
Partitions or swimlanes, activities may involve different participants



Connectors help untangle your diagrams, connecting edges with symbols instead of explicit lines.



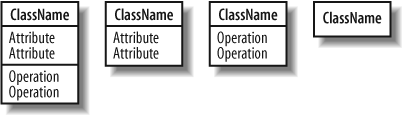
Expansion regions show that actions in a region are performed for each item in an input collection



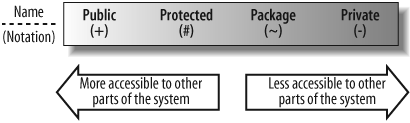
## Modeling a system’s logical structure introducing classes and class diagrams

Abstraction

Encapsulation



Visibility



Multiplicity allows you specify that an attribute represents a collection of objects

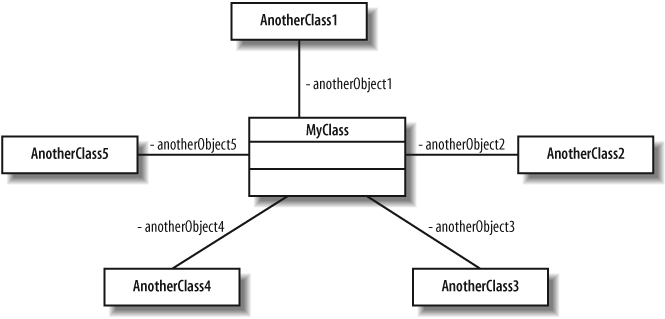


Readonly property applied

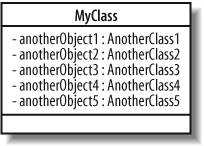


### Inline attributes versus attributes by association

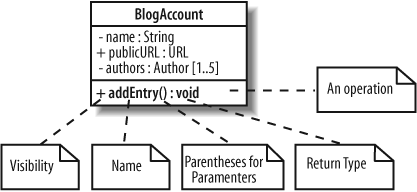
Association



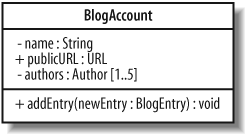
Inline



### Class behavior operations

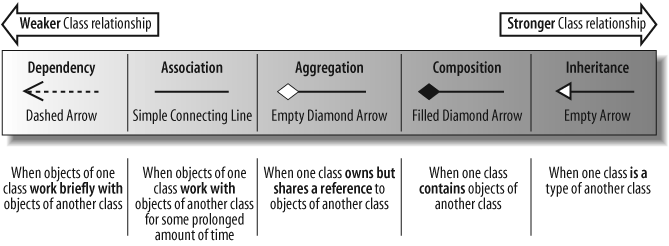


Parameters and return type

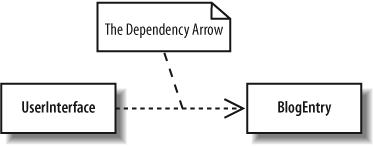


## Modeling a system’s logical structure

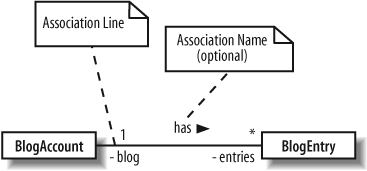
Class relationships

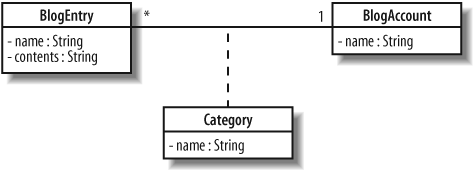


Dependency

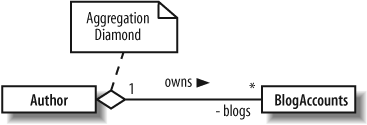


Association

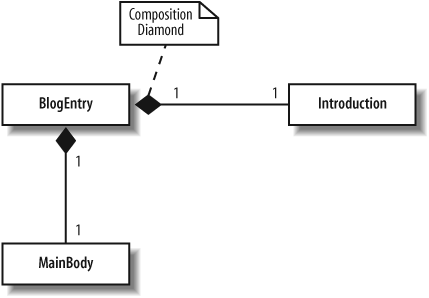




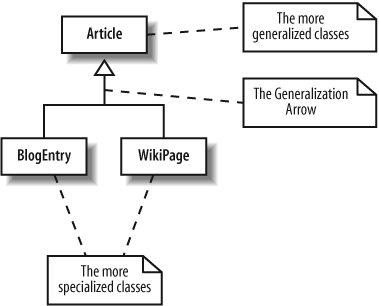
Aggregation, a class owns but may share objects of another class



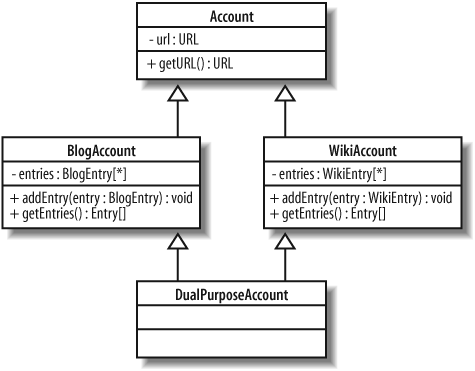
Composition, stronger relationship than aggregation



Generalization, otherwise known as inheritance



Multiple inheritance

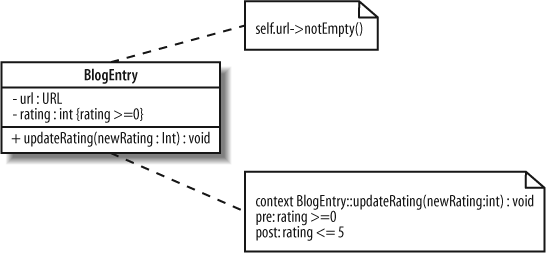


Constraints

Invariants, a constraint that must always be true

Preconditions, is a constraint that is defined on a method and is checked before the method executes

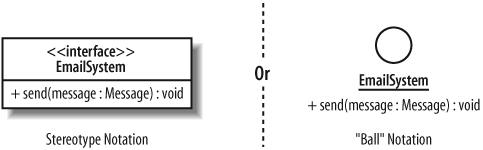
Postconditions, defined on a method, is check after the method executes

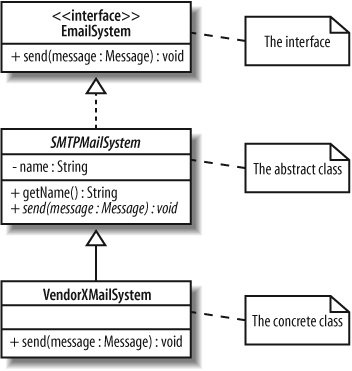


Abstract class

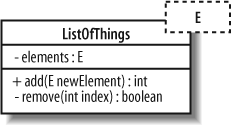


Interface





Templates, parameterized with the type referred to as E. There is no class in our model called E



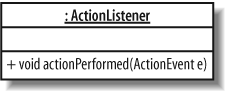
# Bring your classes to life, object diagrams

It helps to capture the logical view of your model

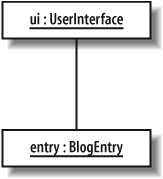
Object instances

The entry object is an instance of the BlogEntry class from Chapter 4

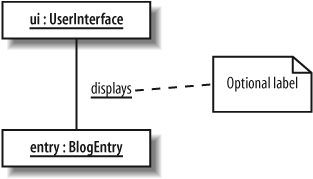
Anonymous object

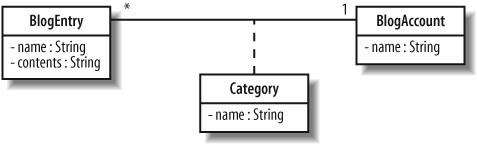


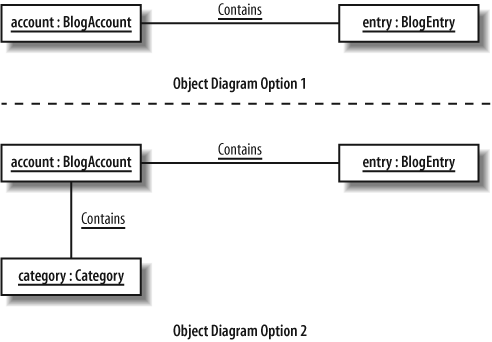
Links, objects on their own are not very interesting or helpful. Work together at runtime configuration



Links and constraints





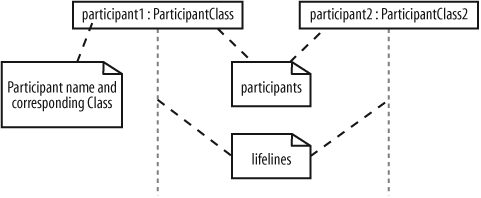


Binding class template

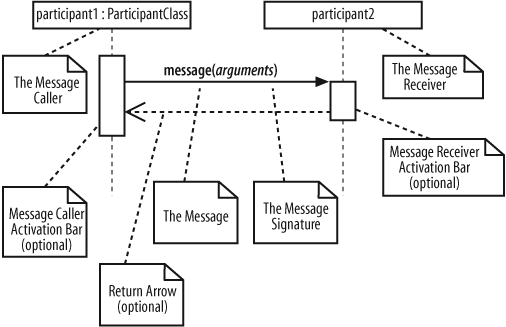
The listOfBlogEntries reuses the generic ListOfThings template, binding the E parameter to the BlogEntry class to store only objects of the BlogEntry class

## Modeling ordered interaction, sequence diagrams

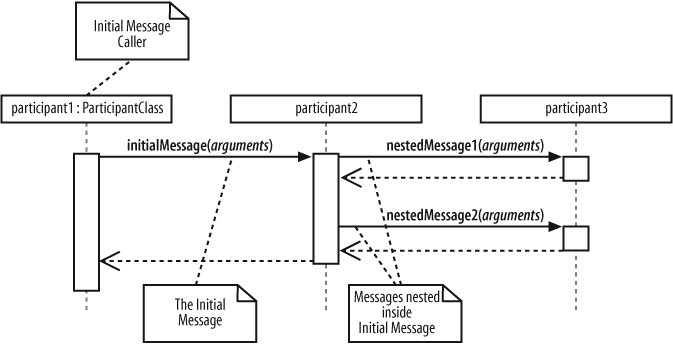
Participants in a sequence diagram



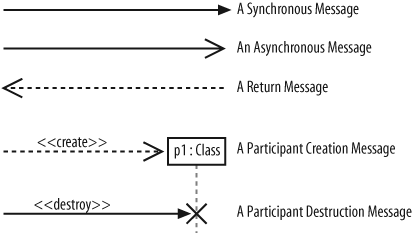
Time, events, signals, activation bars and messages



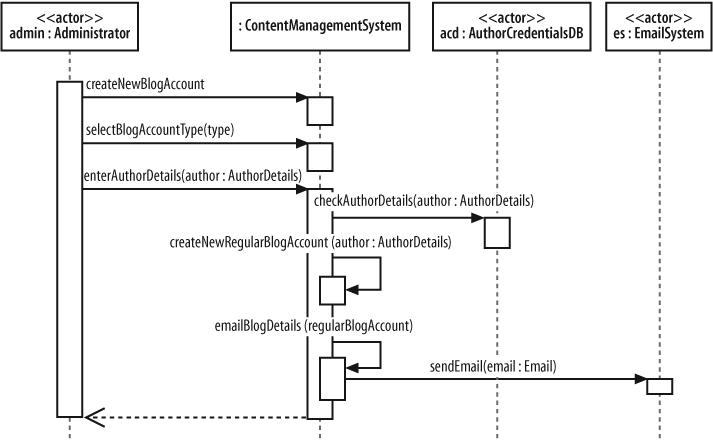
Nested messages

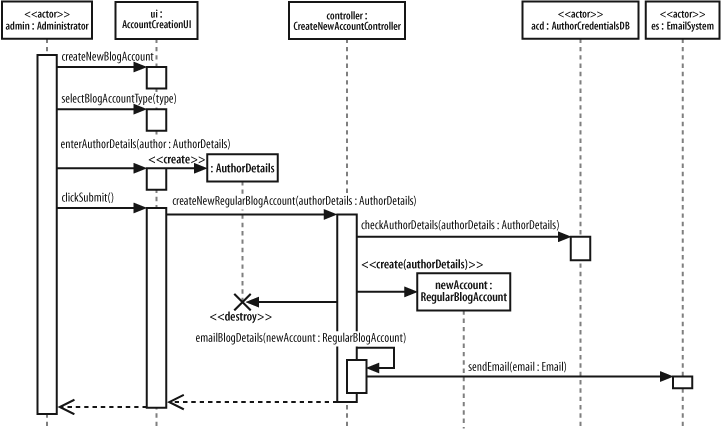


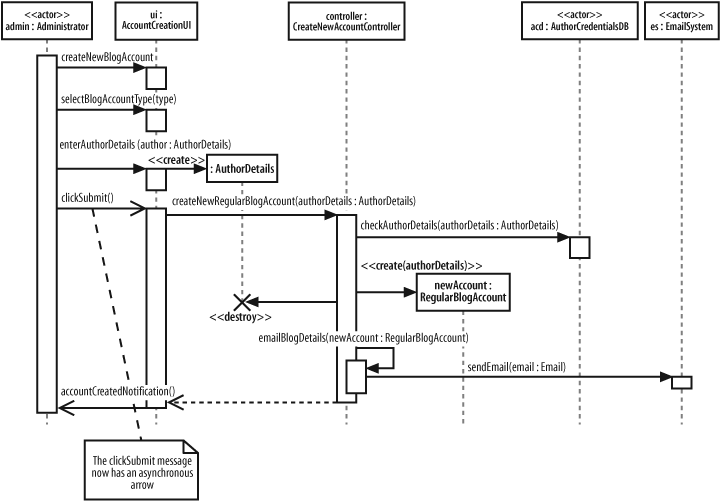
Message arrows



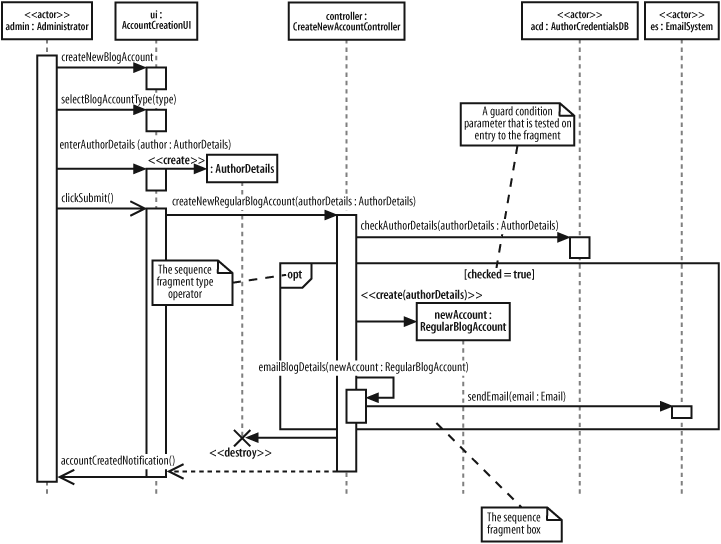
Example a live sequence diagram







Managing complex interactions with sequence fragments, which means that this is an optional fragment. All the interactions contained within the fragment box



UML2.0’s fragment types

Ref , Represents an interaction that is defined elsewhere in the model

Assert, condition must be occurred exactly as they are indicated

Loop,

Break

Alt, like switch

Opt, execute only if the guard condition evaluates to true

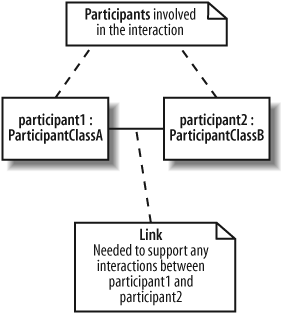
Neg, declares that the interactions inside this fragment are not to be executed

Par, specifies that interactions within this fragment can happily execute in parallel.

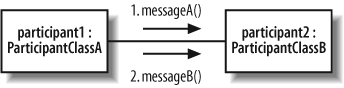
Region, Interactions within this type of fragment are said to be part of a critical region

## Focusing on interaction links

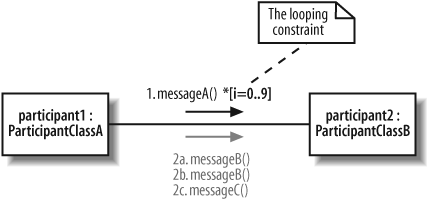
Participants, links and messages



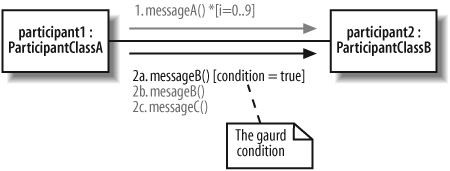
A message on a communication diagram



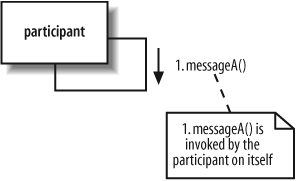
Invoke message multiple times



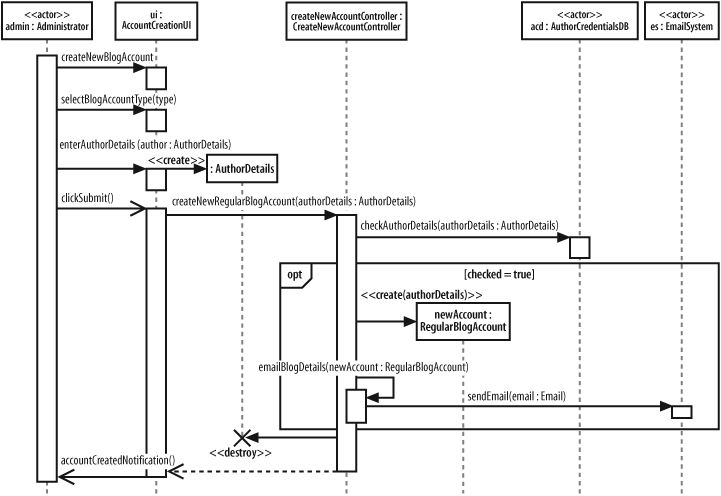
Add condition

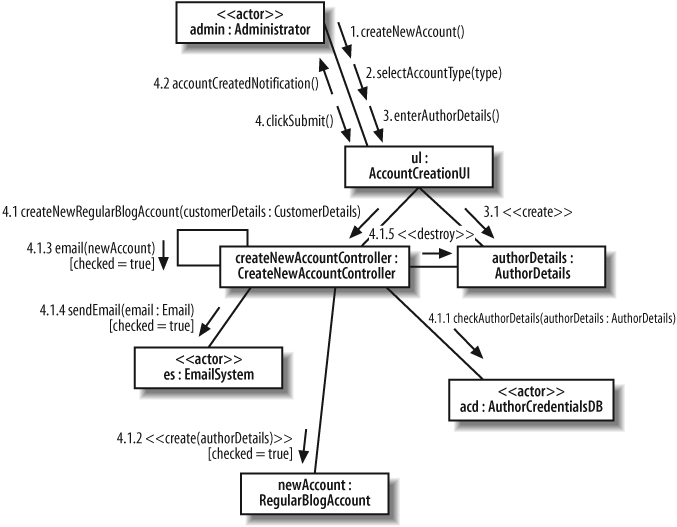


Message to a participant itself



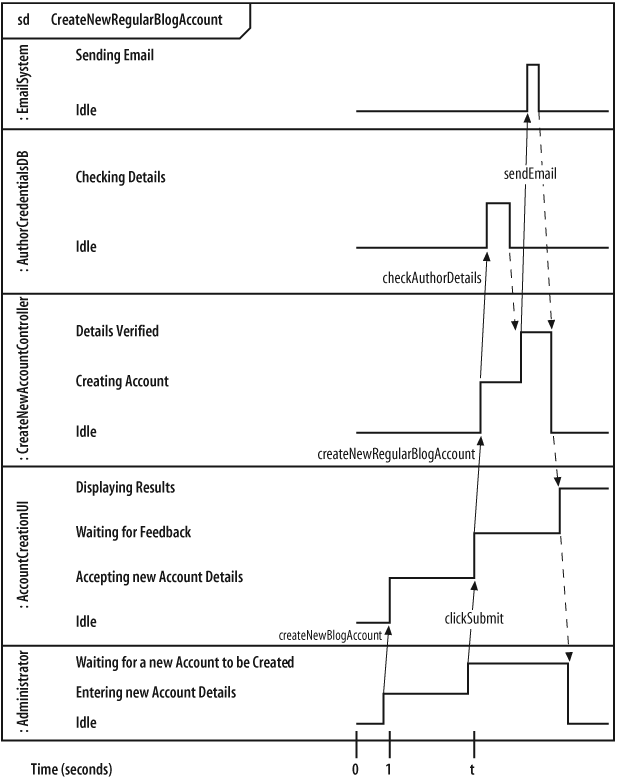
Fleshing out an interaction with a communication diagram, example. Compare communication and sequence diagram





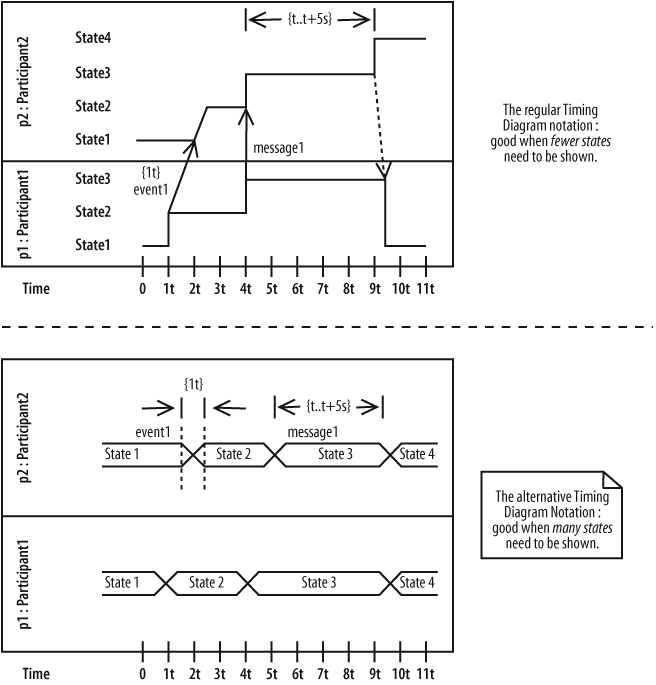
## Focusing on interaction timing, timing diagram

### Participations, states, timeline, constraint



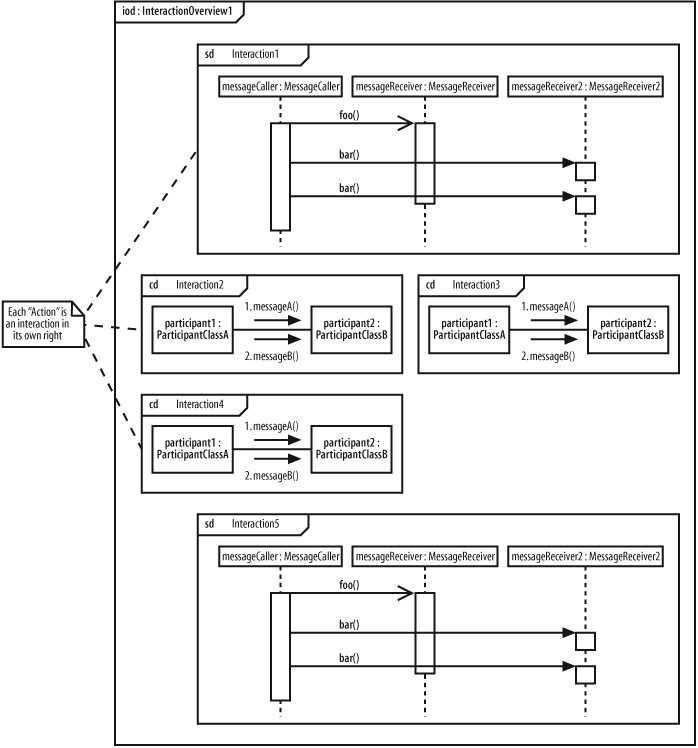
Time diagram can be converted from communication diagram etc.

An alternative notation

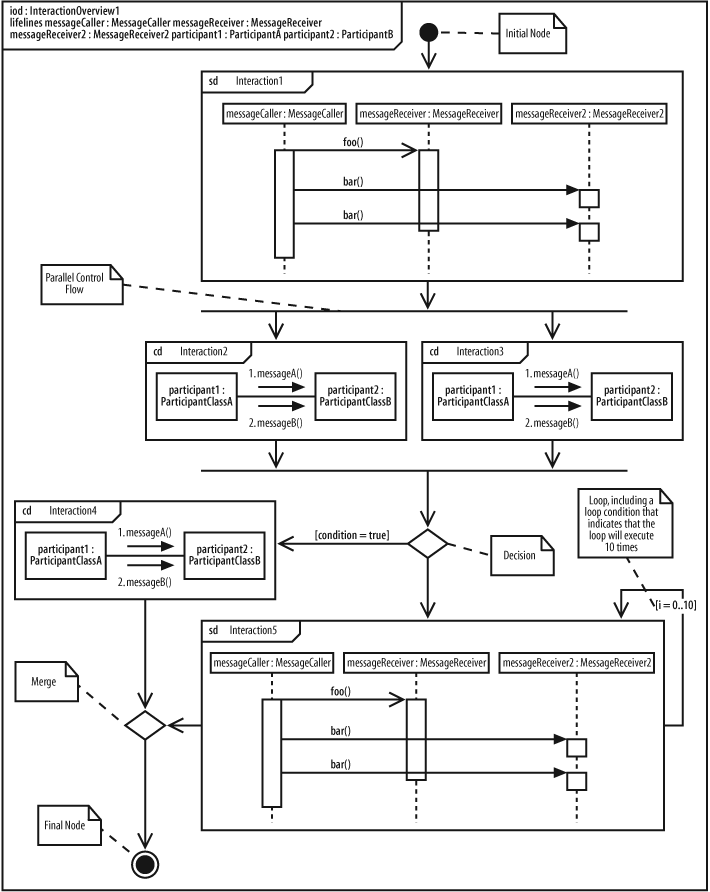


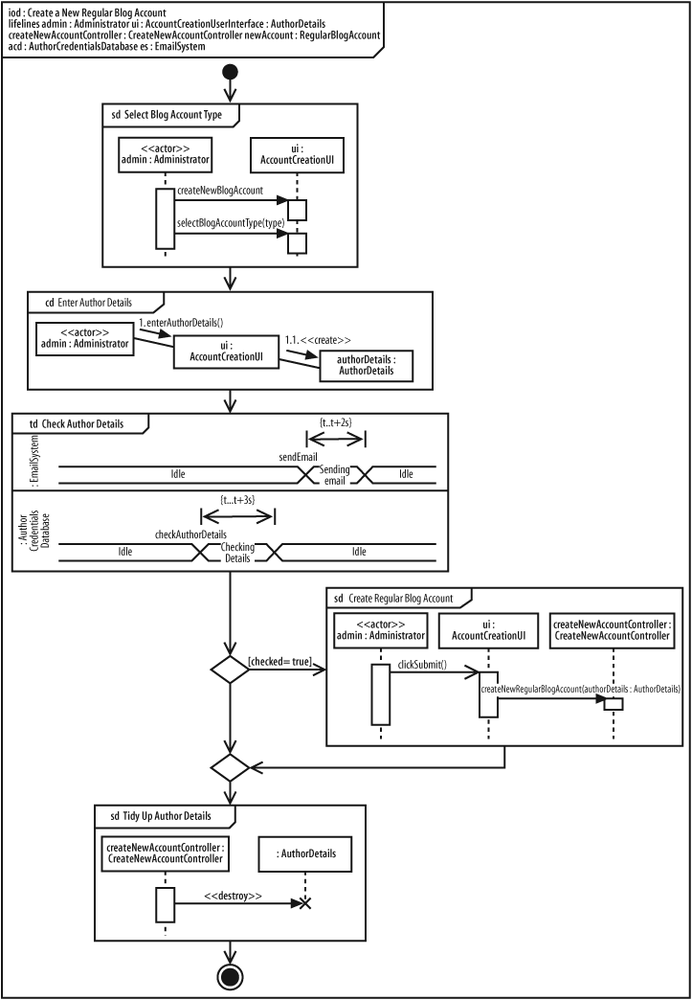
## Completing the interaction picture, interaction overview diagrams

Parts of an interaction overview

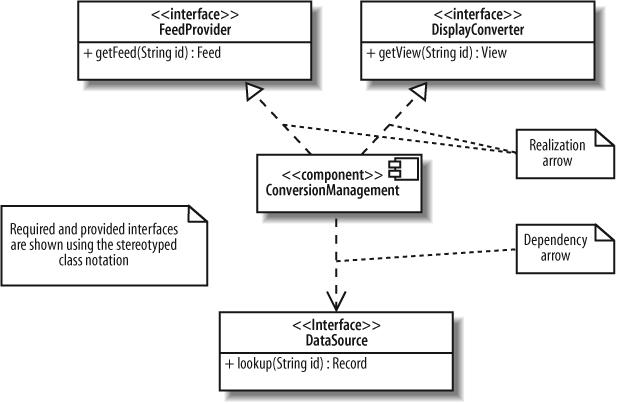


Modeling a use case using an interaction overview

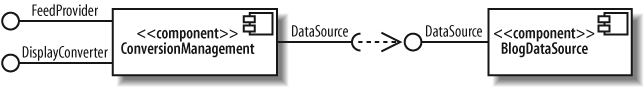




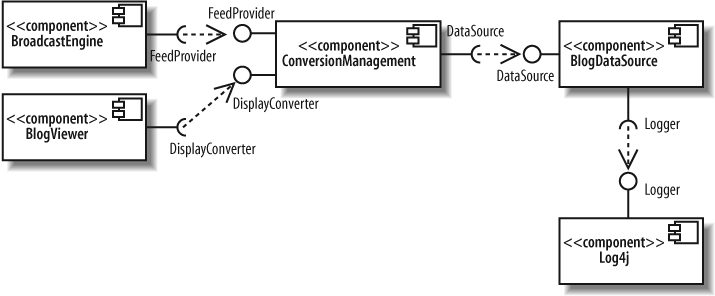
Managing and reusing your system’s parts



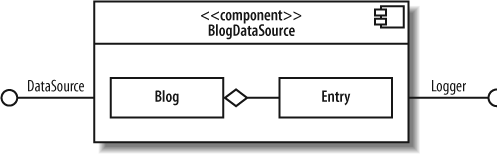
Showing components working together



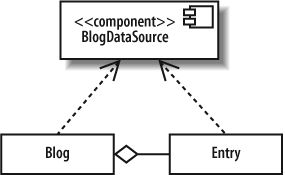
Focusing on the key components and interfaces in your system



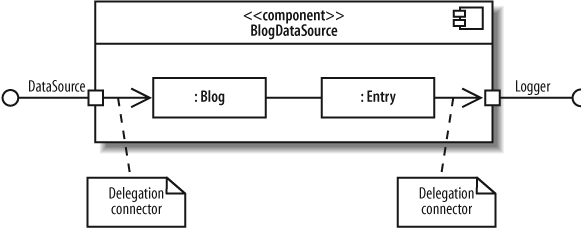
Classes that realize a component



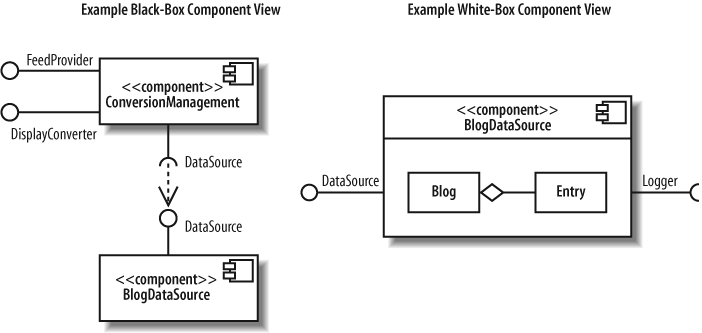
Alternate view, showing the realizing classes



Delegation connectors show how interfaces correspond to internal parts

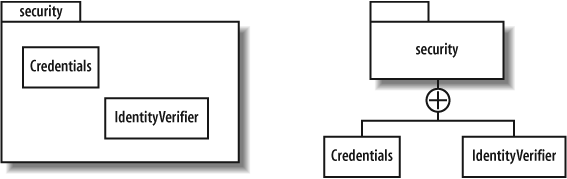


Black box and white box component view

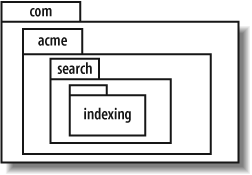


## Organize your model

### Packages



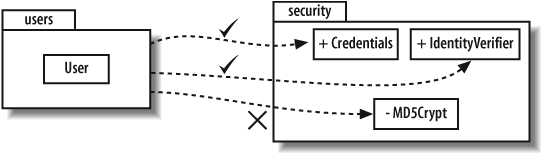
Nested packages



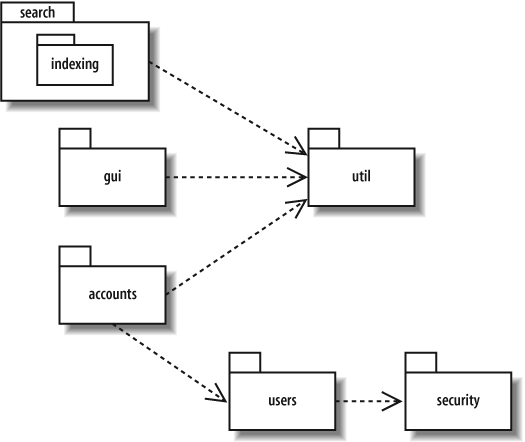
Namespace reference each other

Representing a class with its fully-scoped name: both the security and utils packages have a class named Credentials

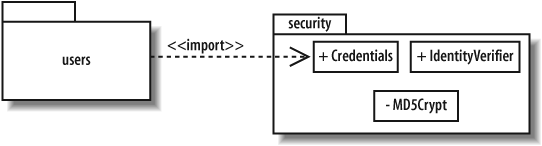
Element visibility



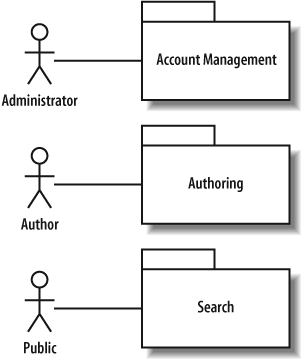
Package dependency



Importing packages

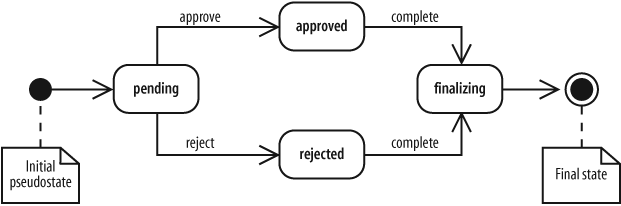


Packages diagram to organize use case

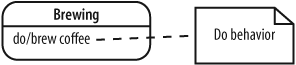


## Modeling an object’s state, state machine diagram

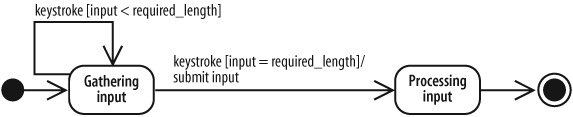
### Essentials



States

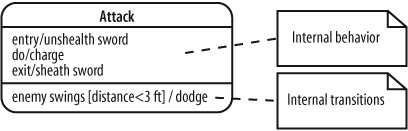


### Transitions, represents a change of states from a source state to a target state . A transition description, written along the arrow



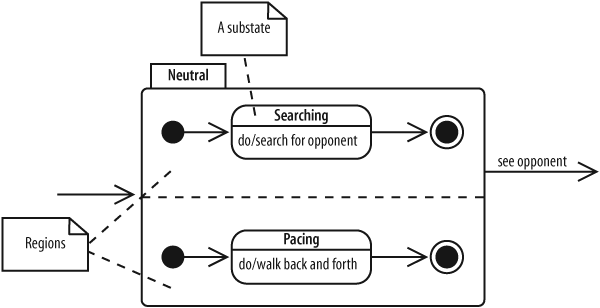
### Advanced state behavior

Internal state

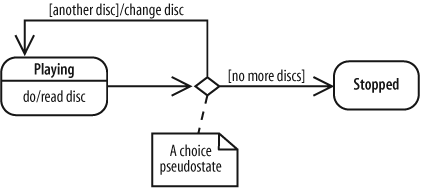


### Composite states

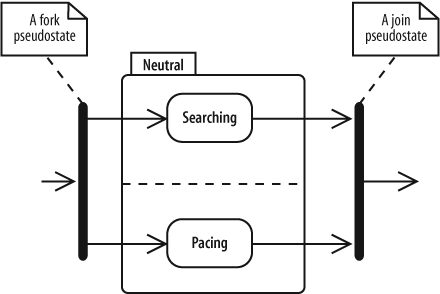
UML allows concurrent states, or being in multiple states at the same time.



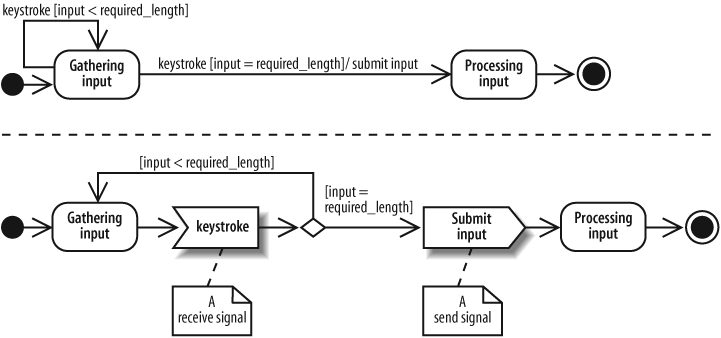
Advanced pseudo states



Forks and joins concurrent states

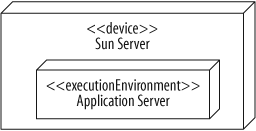


Signals

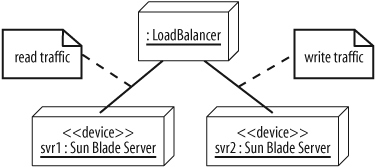


## Deployed system diagrams

Hardware execution environment



A node is a hardware or software resource that can host software or related files.



Communication between nodes

A Desktop PC and Server communicate via TCP/IP

Deployment specification



Provide physical design of the system

