Introduction to Rational Rose

Speaker : Yen Cheng Lai

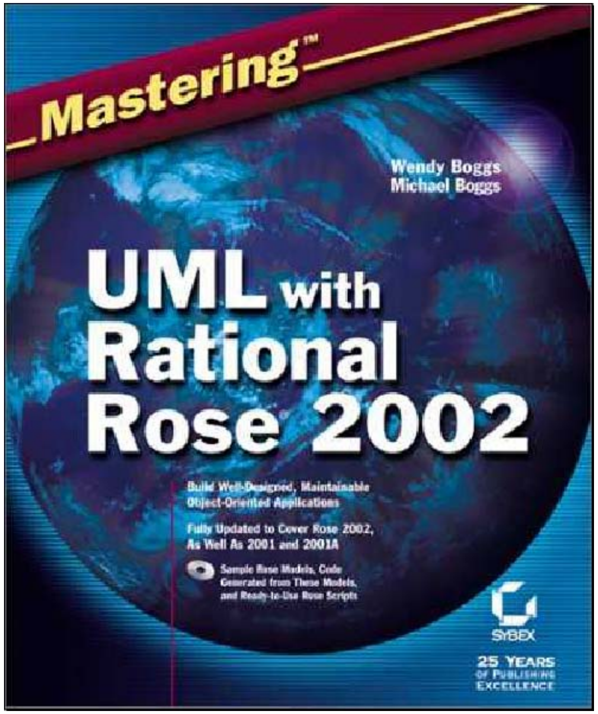
Advisor : Phone Lin

Outline

- Reference
- Rational Rose Setup
- Introduction to Rational Rose
- Use Case view in Rational Rose
- Logical view in Rational Rose
- Component view in Rational Rose
- Deployment view in Rational Rose

Reference

- Book
  - **Wendy Boggs, and Michael Boggs. "Mastering UML with Rational Rose 2002", SYBEX Inc., 2002**



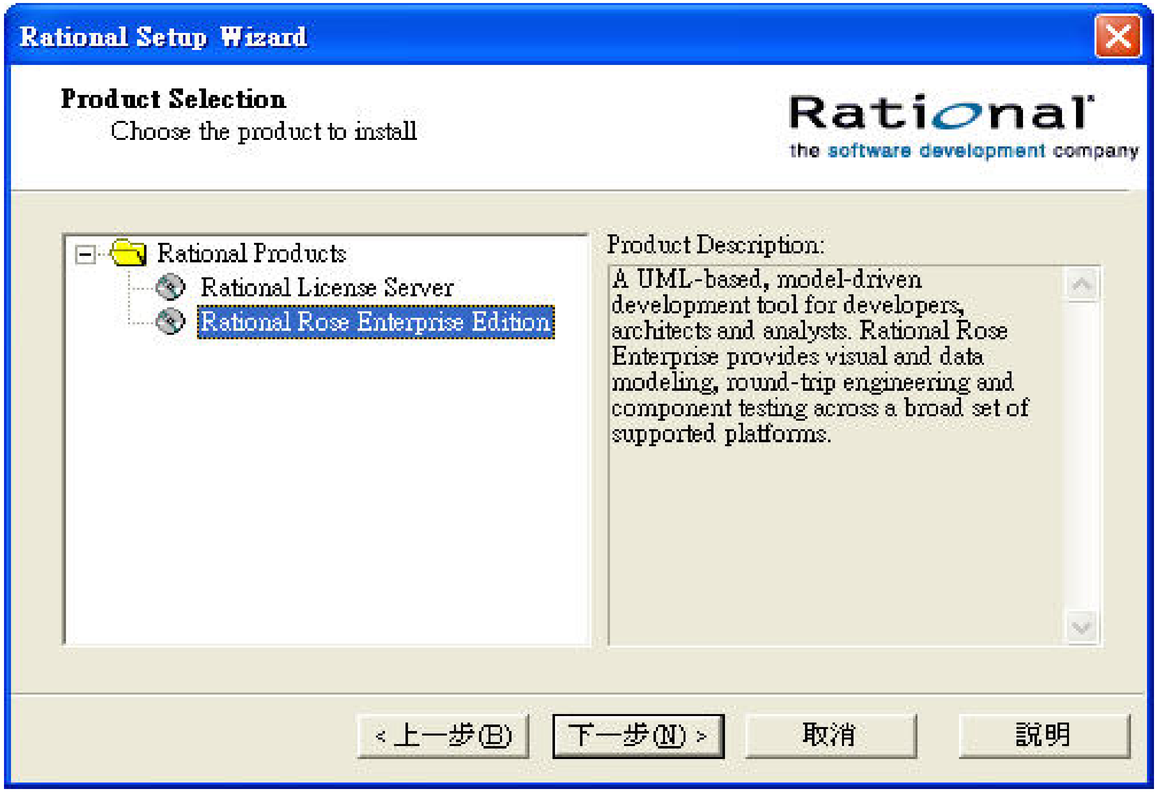
Reference (cont.)

Rational Rose Trial Version

- Rational Rose Enterprise Evaluation Assembly V2003.06.00 for Windows 2000, NT, XP
- Rational Rose Enterprise 15 days license key
- Web Site : http://www14.software.ibm.com/webapp/download/preconfig.jsp?id=2003-09-18+15%3A01%3A34.295232R&S_TACT=TrialsAndBetas&S_CMP=&s=p

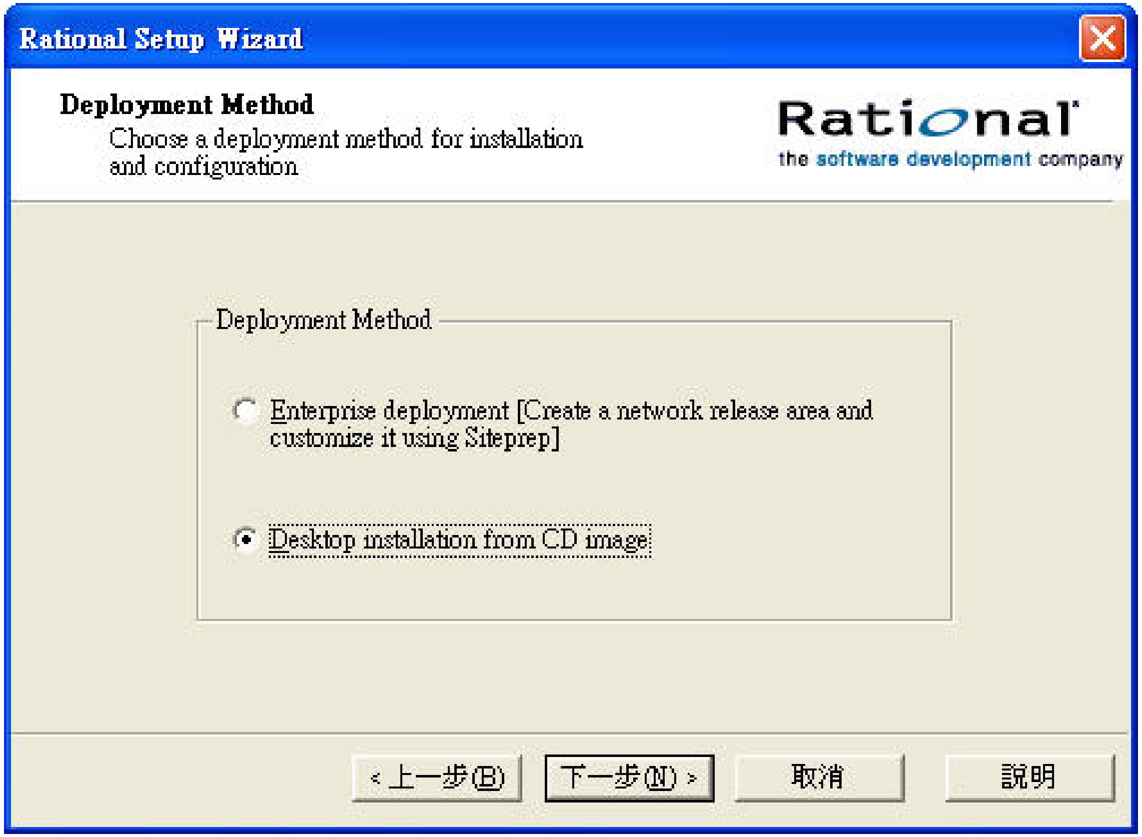
Setup

⌘ Step 1 : Execute setup file and select rational rose enterprise edition



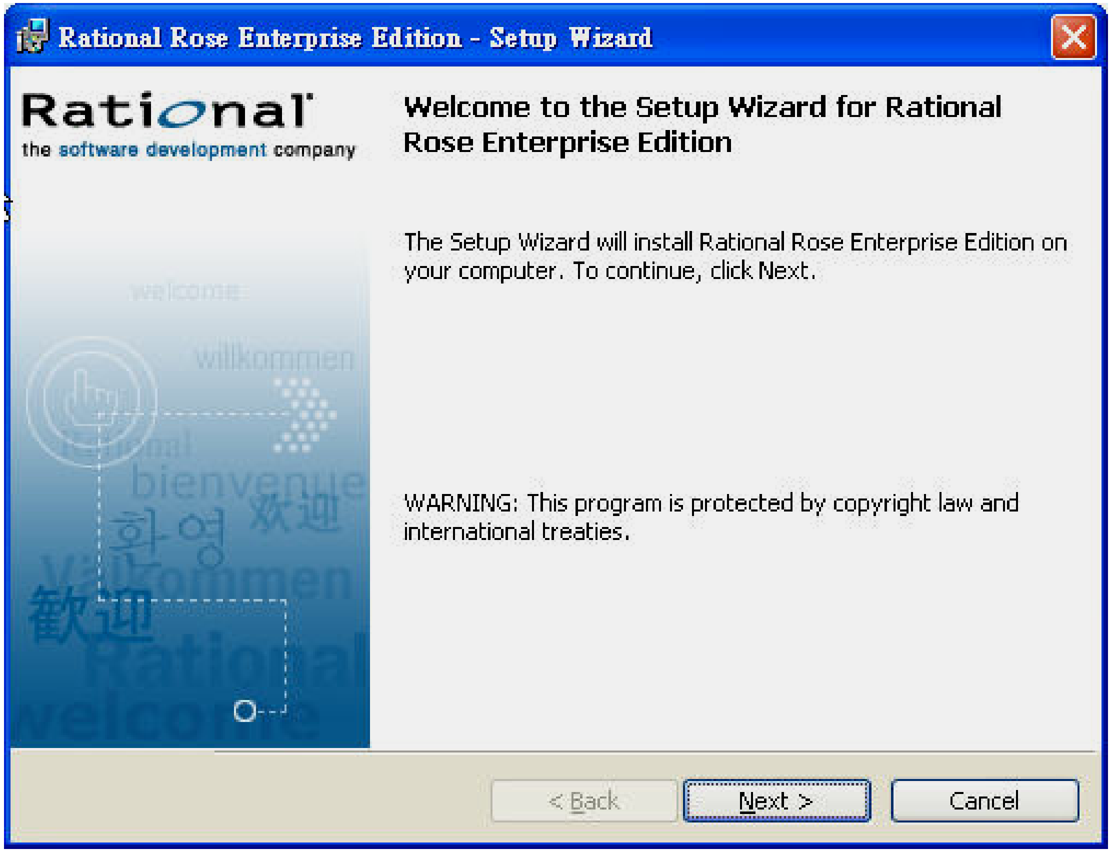
Setup (cont.)

⌘ Step 2 : Select desktop installation from CD image



Setup (cont.)

- Step 3 : Start to install the rational rose

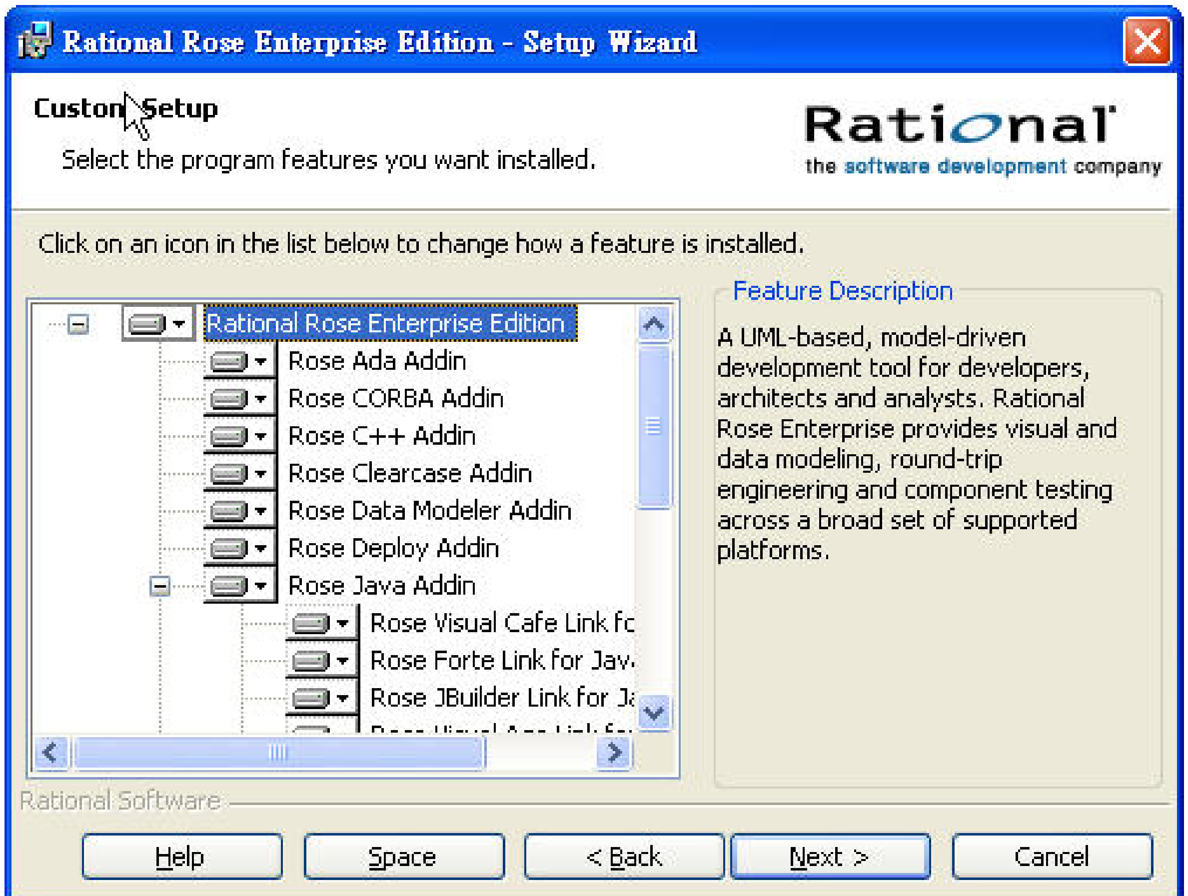


Setup (cont.)

- Step 4

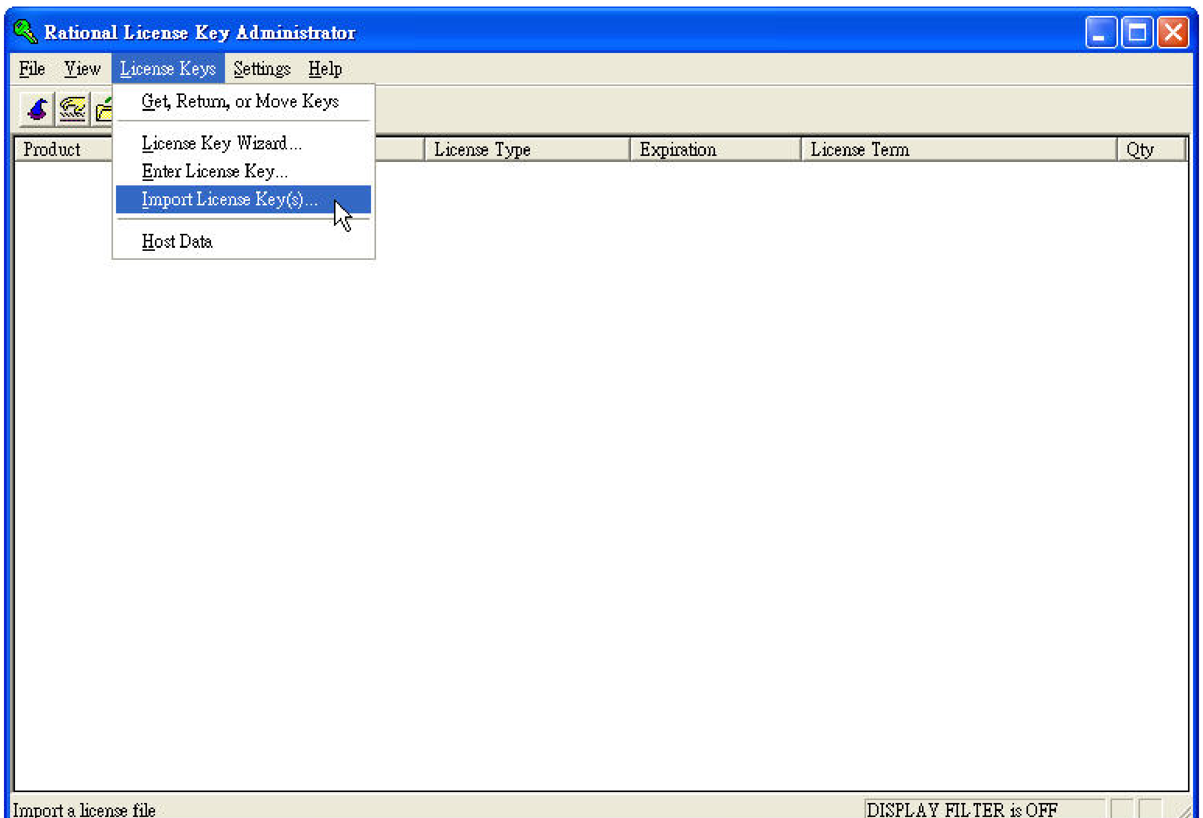
Setup (cont.)

- Step 5



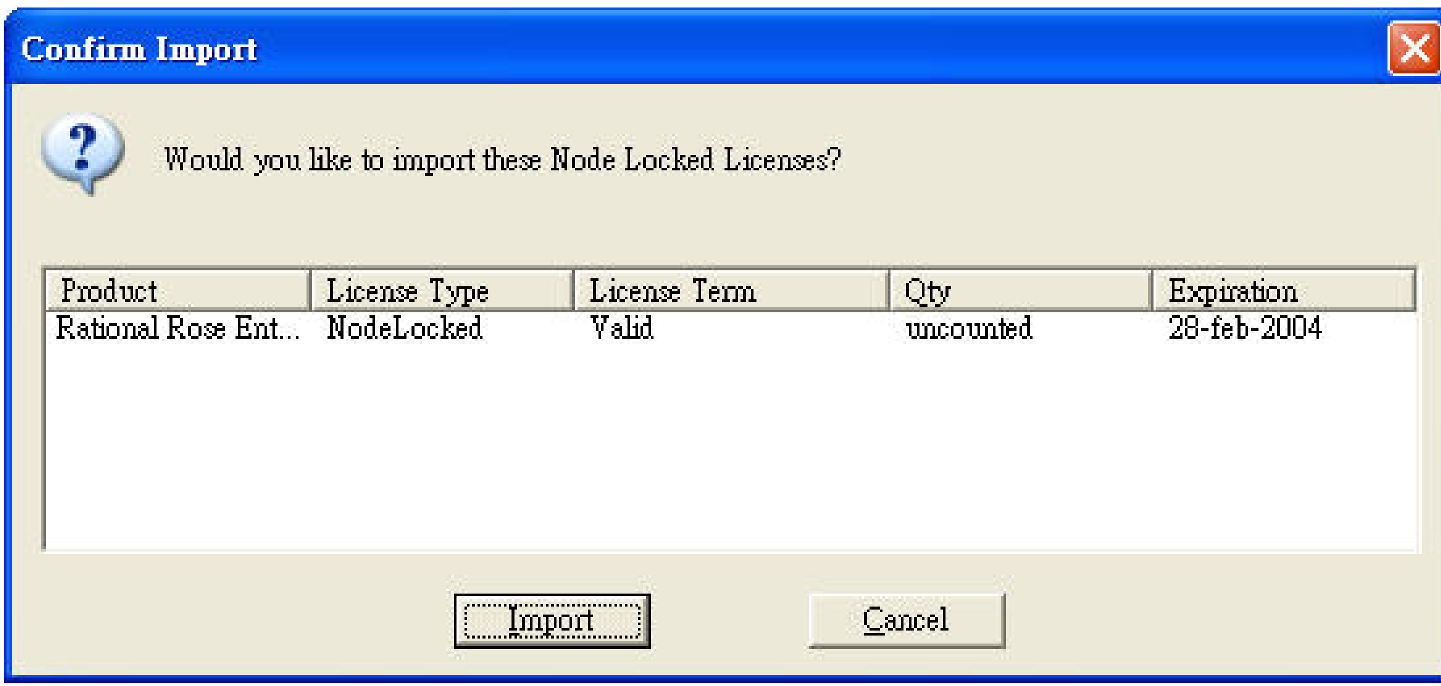
Setup (cont.)

- Step 6 : After installation, we must import license key to activate rational rose.

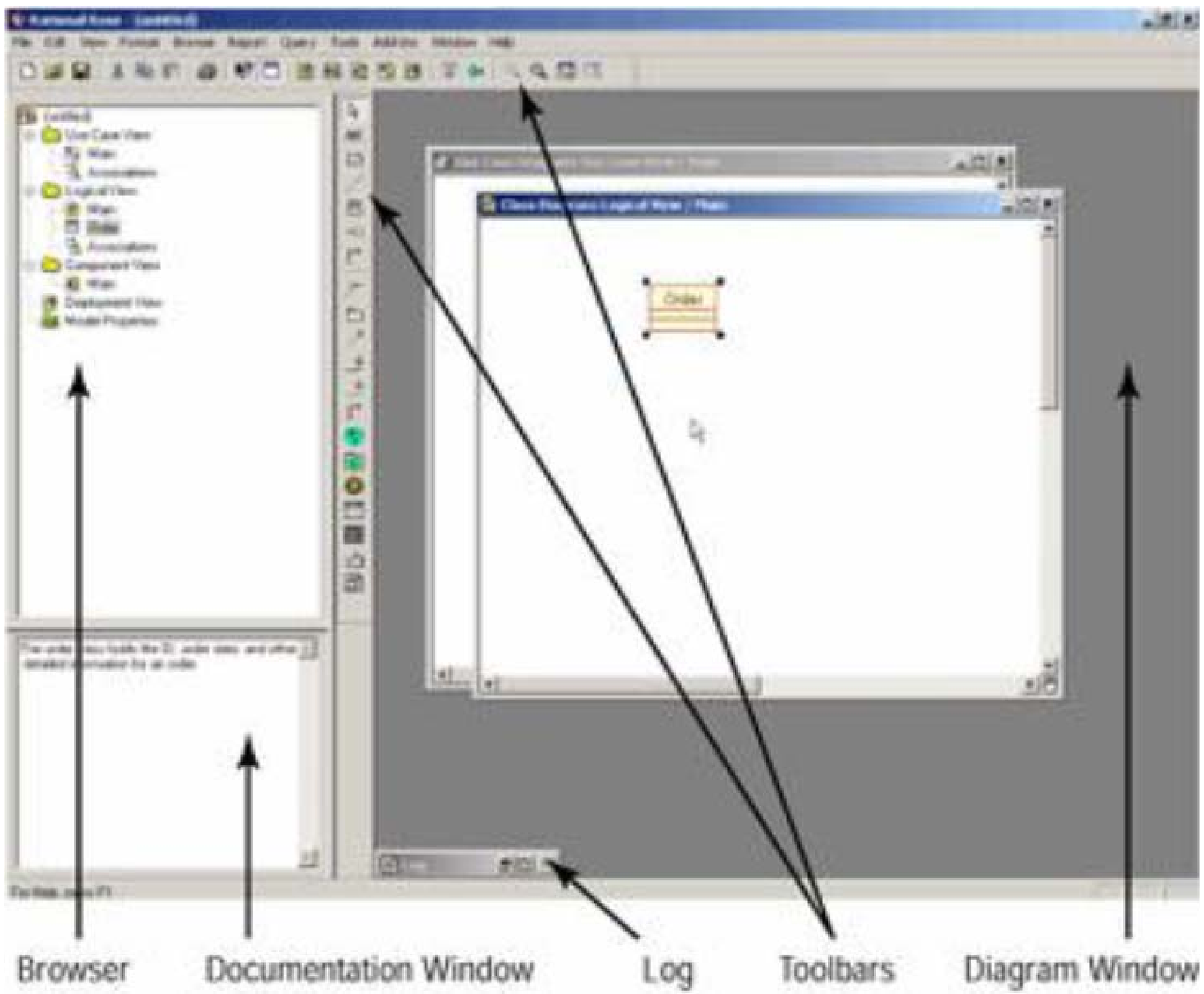


Setup (cont.)

- Step 7



Setup (cont.)


Browser
Documentation Window
Log
Toolbars
Diagram Window

Setup (cont.)

	Cut	Moves text to the clipboard.
	Copy	Copies text to the clipboard.
	Paste	Pastes text from the clipboard.
	Print Diagrams	Prints one or more diagrams from the current model.
	Context Sensitive Help	Accesses the help file.
	View Documentation	Views the documentation window.
	Browse Class Diagram	Locates and opens a Class diagram.
	Browse Interaction Diagram	Locates and opens a Sequence or Collaboration diagram.
	Browse Component Diagram	Locates and opens a Component diagram.
	Browse State Machine Diagram	Locates and opens a Statechart diagram.
	Browse Deployment Diagram	Opens the Deployment diagram for the model.
	Browse Parent	Opens a diagram's parent diagram.
	Browse Previous Diagram	Opens the diagram you were most recently viewing.
	Zoom In	Increases the zoom.
	Zoom Out	Decreases the zoom.
	Fit in Window	Sets the zoom so the entire diagram fits within the window.
	Undo Fit in Window	Undoes the Fit in Window command.

Introduction

- Rational Rose is a powerful visual modeling tool to aid in the analysis and design of object-oriented software systems.
- It is used to model your system before you write any code, so you can be sure that the system is architecturally sound from the beginning.
- It helps with system analysis by enabling you to design use cases and Use Case diagrams to show the system functionalities.

Introduction (cont.)

- It will let you design Interaction diagram to show how the objects work together to provide the needed functions.
- Class diagrams can be created to show the classes in a system and how they relate to each other.
- Component diagrams can be developed to illustrate how the classes map to implementation components.
- Finally, Deployment diagrams can be produced to show the network design for the system.

Introduction (cont.)

- Customer and project managers will use the Use Case diagrams to get a high-level view of the system and to agree on the project score.
- Project managers will use the Use Case diagrams and documents to break the project down into manageable pieces.
- Analysts and customers will look at the use case document to see what functionality the system will provide.

Introduction (cont.)

- Technical writers will look at the use case document to begin to write the user manual and training plans.
- Analysts and developers will look at Sequence and Collaboration diagrams to see how the logic in the system will follow, the objects in the system, and the messages between the objects.
- Quality assurance staff will use the use case document and the Sequence and Collaboration diagrams to get the information they need for testing scripts.

Introduction (cont.)

- Developers will use the Class diagrams and Statechart diagrams to get a detailed view of the pieces of the system and how they relate.
- Deployment staff will use the Component and Deployment diagrams to see what executable files, DLL files, or other components will be created.
- The whole team will use the model to be sure the requirements are traced to the code, and that the code can be traced back to the requirements.

Introduction (cont.)

- There are four views in Rational Rose
  - Use Case view
  - Logical view
  - Component view
  - Deployment view

Use Case view

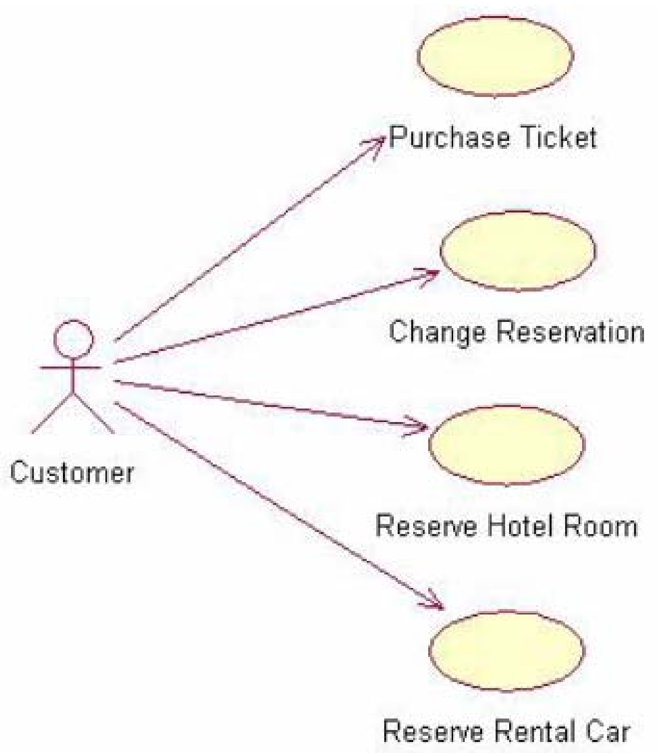
- The Use Case view is an implementation-independent look at the system.
- It focuses on a high-level picture of what the system will do, without worrying about the details of how the system will do.
- Use Case view includes:
  - Actors
  - Use cases
  - Associations
  - Use case documentation
  - Use Case diagrams
  - Sequence diagrams
  - Collaboration diagrams
  - Packages

Use Case Diagram

- A Use Case diagram is a graphical representation of the high-level system scope.
- It includes *use cases*, which are pieces of functions the system will provide, and *actors*, who are the users of the system.
- Looking at a Use Case diagram, you should easily be able to tell what the system will do and who will interact with it.
- To create a new Use Case diagram:
  - 1. Right-click a package in the Use Case view.
  - 2. Select New → Use Case Diagram.

Use Case Diagram (cont.)

- A *Use Case diagram* shows a subset of the use cases and actors in the system. You can create as many Use Case diagrams as you need to fully document the system scope.



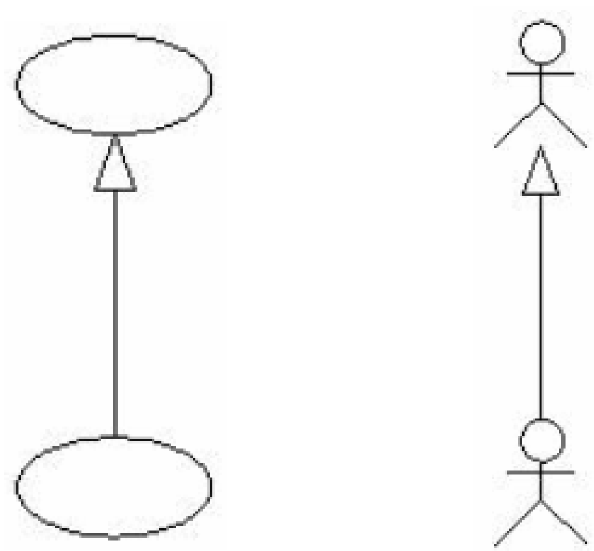
Use Case Diagram (cont.)

- An *includes relationship* suggests that one use case must include another. In other words, running one use case means that the other must be run as well. One use case may be included by one or more other use cases.

- An *extends relationship* is used when one use case optionally extends the functionality provided by another. In other words, if one use case runs, an extending use case may or may not run.

Use Case Diagram (cont.)

- A *generalization relationship* between two use cases indicates that one use case (the child) inherits all of the functionality provided by the other use case (the parent).



Use Case Diagram (cont.)

ABC	Text Box	Adds a text box to the diagram.
	Note	Adds a note to the diagram.
	Anchor Note to Item	Connects a note to a use case or actor on the diagram.
	Package	Adds a new package to the diagram.
	Use Case	Adds a new use case to the diagram.
	Actor	Adds a new actor to the diagram.
	Unidirectional Association	Draws a relationship between an actor and a use case.
	Dependency or Instantiates	Draws a dependency between items on the diagram.
	Generalization	Draws a includes or an extends relationship between use cases, or draws an inheritance relationship between actors.

Interaction Diagram

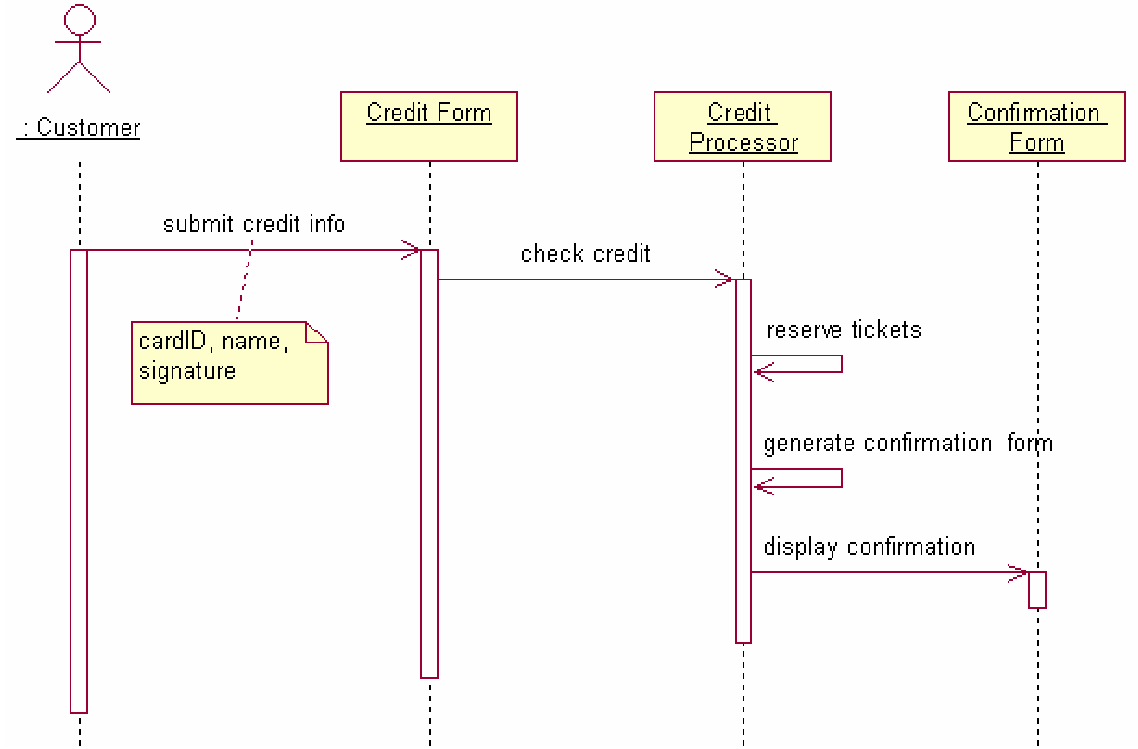
- An Interaction diagram is a graphical representation of how the objects and actors in a system interact with one another to achieve the desired goal of the system.
- There are two types of Interaction diagrams, *Sequence diagrams* and *Collaboration diagrams*.
- Sequence diagrams illustrate the interactions of objects along a timeline.
- Collaboration diagrams show the interactions, but without the timeline.
- These two diagrams display the same information, just in different ways.

Interaction Diagram (cont.)

- Sequence diagrams are usually created to show the flow of functionality and control throughout the objects in the system.
- Collaboration diagrams are typically used to illustrate which objects communicate with other objects.
- Sequence diagrams answer questions about how the system will work.
- Collaboration diagrams answer questions about the soundness of the structure of the system.

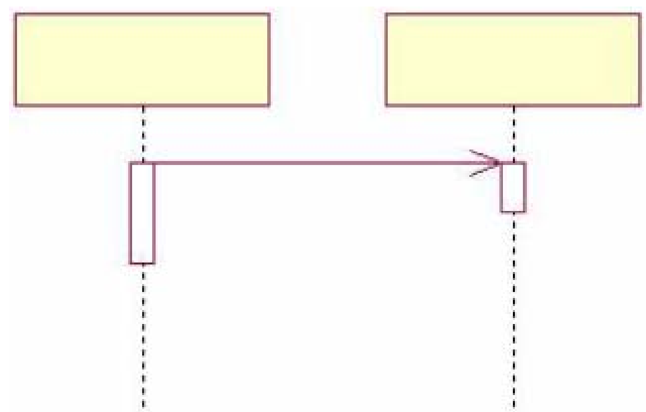
Sequence Diagram

- To create a new Sequence diagram:
  - 1. Right-click a use case in the Use Case view.
  - 2. Select New -> Sequence Diagram.

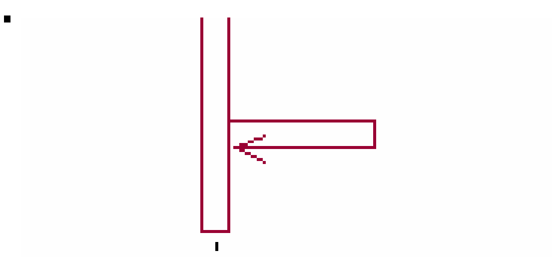


Sequence Diagram (cont.)

- A message is simply some form of communication between one object or actor and another.



- Messages can also be *reflexive*, meaning that the object communicates some information to itself.



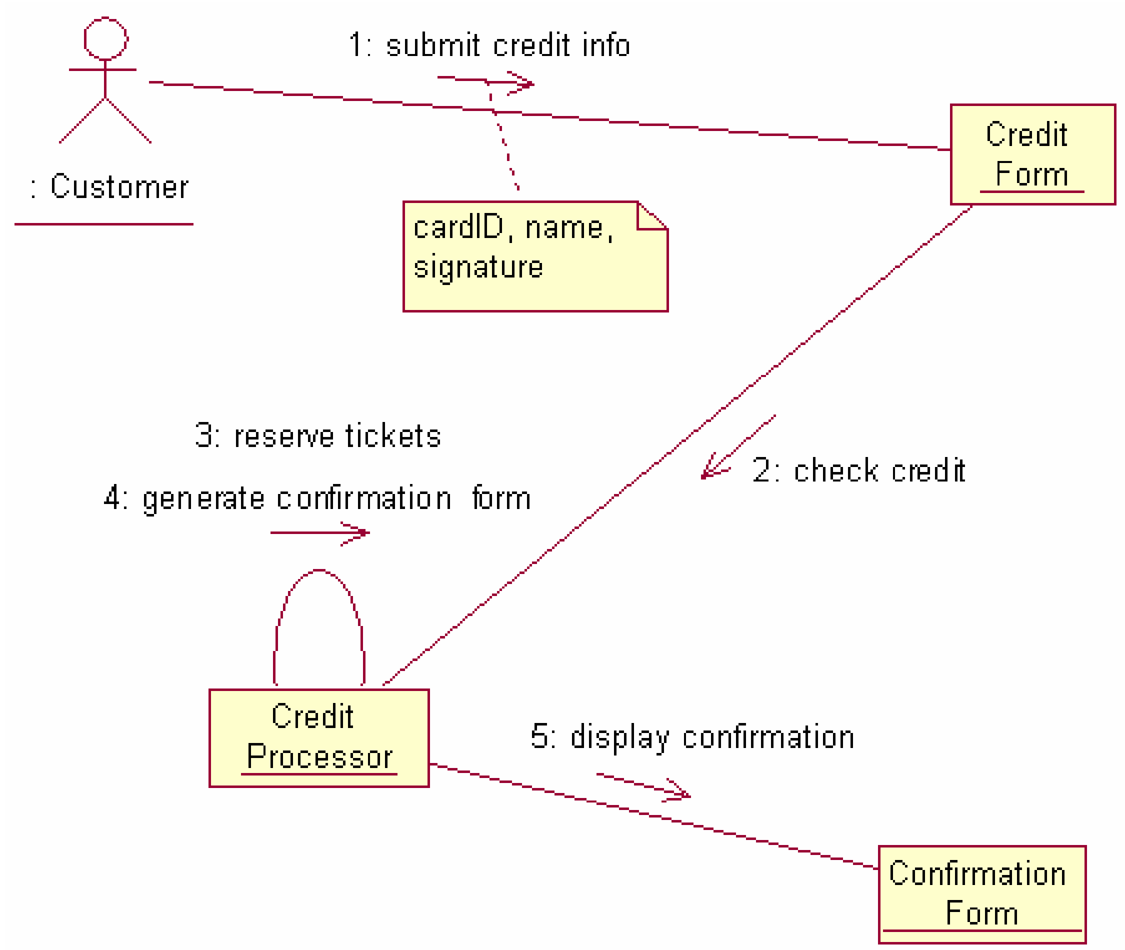
Sequence Diagram (cont.)

	Text Box	Adds a text box to the diagram.
	Note	Adds a note to the diagram.
	Anchor Note to Item	Connects a note to an item in the diagram.
	Object	Adds a new object to the diagram.
	Object Message	Draws a message between two objects.
	Message to Self	Draws a reflexive message.
	Return Message	Shows a return from a procedure call.
	Destruction Marker	Shows when an object is destroyed.
	Procedure Call	Draws a procedure call between two objects.
	Asynchronous Message	Draws an asynchronous message between two objects.

Collaboration Diagram

- Collaboration diagrams are not time-based, but show the interactions of the objects as a whole.
- This is useful for pointing out potential bottlenecks (objects that individually communicate with a large number of objects).
- Sequencing of messages is shown on a Collaboration diagram by numbering the messages.
- To create a new Collaboration diagram:
  - 1. Right-click a use case in the Use Case view.
  - 2. Select New → Collaboration Diagram.

Collaboration Diagram (cont.)



Collaboration Diagram (cont.)

Icon	Button	Purpose
	Selects or Deselects an Item	Returns the cursor to an arrow to select an item.
	Text Box	Adds a text box to the diagram.
	Note	Adds a note to the diagram.
	Anchor Note to Item	Connects a note to an item on the diagram.
	Object	Adds a new object to the diagram.
	Class Instance	Adds a new class instance to the diagram.
	Object Link	Creates a path for communication between two objects.
	Link to Self	Shows that an object can call its own operations.
	Link Message	Adds a message between two objects or from an object to itself.
	Reverse Link Message	Adds a message in the opposite direction between two objects or from an object to itself.
	Data Token	Shows information flow between two objects.
	Reverse Data Token	Shows information flow in the opposite direction between two objects.

Logical view

- The Logical view focuses on how the system will implement the behavior in the use cases.
- It provides a detailed picture of the pieces of the system, and describes how the pieces interrelate. With these detailed elements, developers can construct a detailed design for the system.
- Logical view includes:
  - Classes
  - Class diagrams
  - Associations
  - Interfaces
  - Sequence diagrams
  - Collaboration diagrams
  - Statechart diagrams
  - Packages

Class Diagram

- A *Class diagram* is used to display some of the classes and packages in your system.
- A *Class diagram* includes a subset of the classes, attributes, operations, relationships, and packages of classes in the system.
- It gives you a static picture of the pieces in the system and of the relationships between them.
- Class diagrams help the developers see and plan the structure of the system before the code is written, helping to ensure that the system is well designed from the beginning.

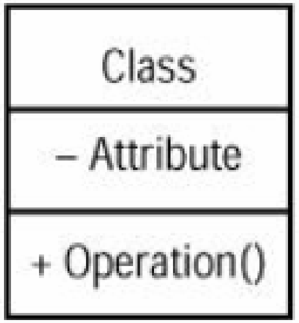
Class Diagram (cont.)

⌘What is a ***Class***?

- A *class* is something that encapsulates information and behavior.
- Traditionally, we've approached systems with the idea that we have the information over here on the database side and the behavior over there on the application side.
- For example, in a personnel system, we may have a class called Employee.
  - This class will contain some information, such as an employee ID, name, address, and phone number.
  - The Employee class will also have some behavior, such as knowing how to hire or fire an employee or giving an employee a raise.

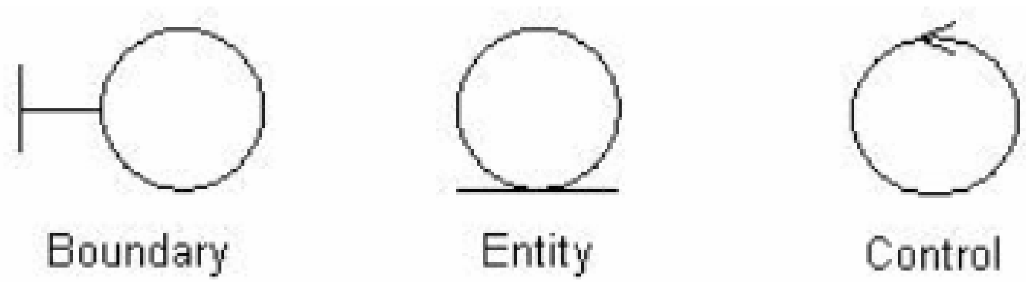
Class Diagram (cont.)

- In UML, a *Class* is shown using the following notation:
  - **Class: holds the class name.**
  - **- Attribute: holds the attributes, or the information that a class holds (such as string, int etc.).**
  - **+ Operations(): holds the operations, or the behavior of a class.**
- If you would like, you can hide the attributes and/or the operations of the class in order to make your diagrams easier to read.



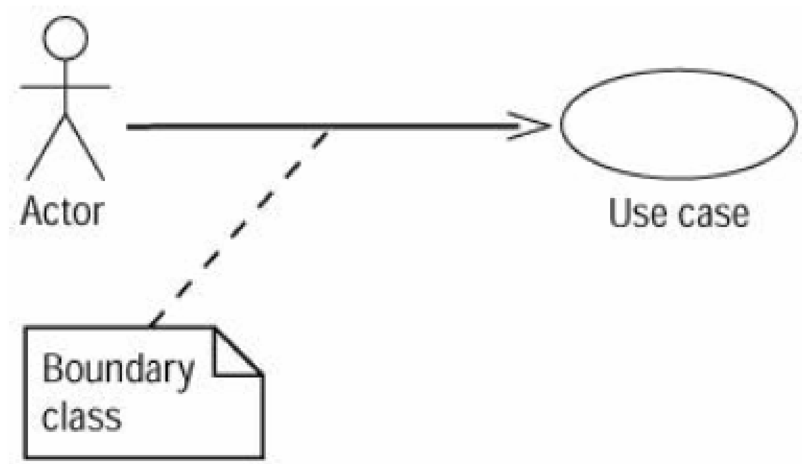
Class Diagram (cont.)

- During analysis, you may want to categorize your classes according to the functions they perform.
- There are three primary class stereotypes in UML that are used for analysis: Boundary, Entity, and Control.



Class Diagram (cont.)

- *Boundary classes* are those classes that lie on the boundary between your system and the rest of the world.
- These would include all of your forms, reports, interfaces to hardware such as printers or scanners, and interfaces to other systems.



Class Diagram (cont.)

- *Entity classes* hold information that you may save to persistent storage.
- Entity classes are usually found in the flow of events and in Interaction diagrams.
- They are the classes that have the most meaning to the user and are typically named using business-domain terminology.

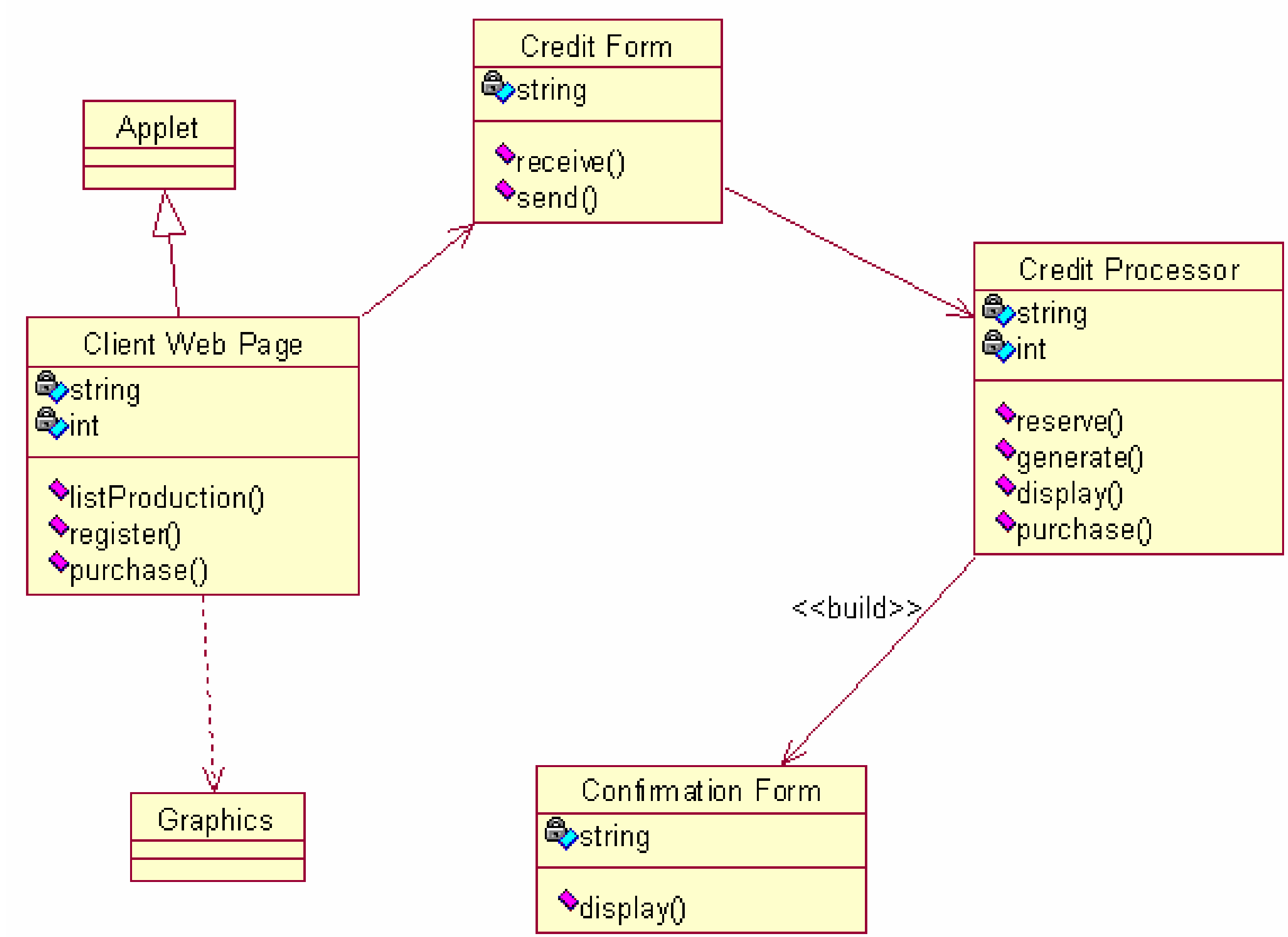
Class Diagram (cont.)

- *Control classes* are responsible for coordinating the efforts of other classes.
- If a control class is used, there is typically one control class per use case, which controls the sequencing of events through the use case.

Class Diagram (cont.)

- Follow these steps to create a new Class diagram:
  - 1. Right-click a package in the Logical view.
  - 2. Select New → Class Diagram.

Class Diagram (cont.)



Class Diagram (cont.)

Icon	Button	Purpose
	Selects or Deselects an Item	Returns the cursor to an arrow to select an item.
	Text Box	Adds a text box to the diagram.
	Note	Adds a note to the diagram.
	Anchor Note to Item	Connects a note to an item on the diagram.
	Class	Adds a new class to the diagram.
	Interface	Adds a new interface class to the diagram.
	Association	Draws an association relationship.
	Aggregation	Draws an aggregation relationship.

Class Diagram (cont.)

	Association Class	Links an association class to an association relationship.
	Package	Adds a new package to the diagram.
	Dependency or Instantiates	Draws a dependency relationship.
	Generalization	Draws a generalization relationship.
	Realize	Draws a realizes relationship.
	Parameterized Class	Adds a new parameterized class to the diagram.
	Class Utility	Adds a new class utility to the diagram.
	Parameterized Class Utility	Adds a new parameterized class utility to the diagram.
	Instantiated Class	Adds a new instantiated class to the diagram.
	Instantiated Class Utility	Adds a new instantiated class utility to the diagram.
	Domain	Adds a new domain to the diagram.
	Domain Package	Adds a new domain package to the diagram.
	Server Page	Adds a new server page to the diagram.
	Client Page	Adds a new client page to the diagram.
	Form	Adds a new HTML form to the diagram.
	COM Object	Adds a new COM object to the diagram.
	Applet	Adds a new applet to the diagram.

Statechart Diagram

- A Statechart diagram shows the life cycle of a single object, from the time that it is created until it is destroyed.
- These diagrams are a good way to model the dynamic behavior of a class.
- A Statechart diagram is used to show the dynamic behavior of an object.
- A *Statechart diagram* includes the various states, transitions, activities, and actions for a particular class.
- Each class can have a single state/activity model, with one or more Statechart diagrams within it.

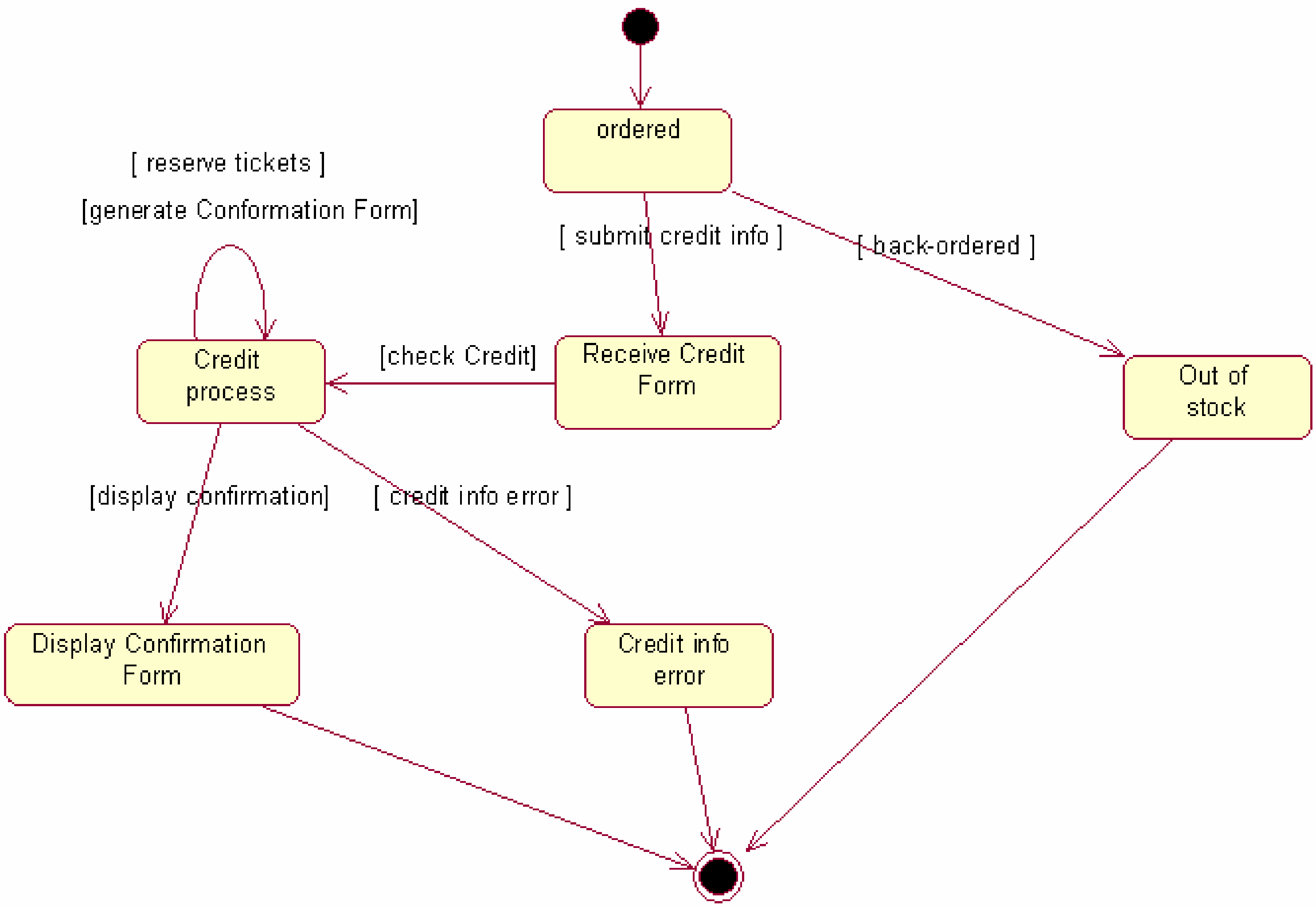
Statechart Diagram (cont.)

- All of this information helps a developer get a complete picture of how a particular object should behave.
- A *state* is a condition in which an object can exist. For example, an invoice can be New, Paid, Delinquent, or Canceled. Each of these represents a different state.

Statechart Diagram (cont.)

- A *start state* shows the state an object is in when it is first initialed. A Statechart diagram has one, and only one, start state.
- An *end state* shows what state an object is in right before it is removed from memory. A Statechart diagram may have zero, one, or more end states.
- A start state is shown as a black dot on the diagram. An end state is shown as a bull's-eye on the diagram.

Statechart Diagram (cont.)



Component view

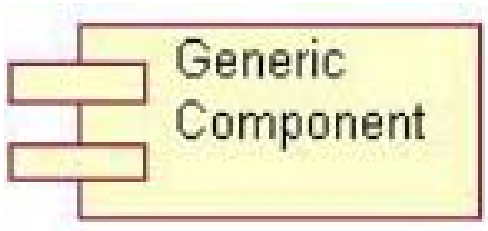
- The Component view contains information about the code libraries, executable files, run-time libraries, and other components in your model.
- The Component view of the system allows you to see the relationships between the modules of code.
- Component view includes:
  - Components
  - Interfaces
  - Component diagrams
  - Packages

Component Diagram

- A *Component diagram* is a UML diagram that displays the components in the system and the dependencies between them.
- The staff responsible for compiling and deploying the system will know which code libraries exist and which executable files will be created when the code is compiled.
- Developers will know which code libraries exist and what the relationships are between them.
- The component dependencies will let those who are responsible for compiling know in which order the components need to be compiled.

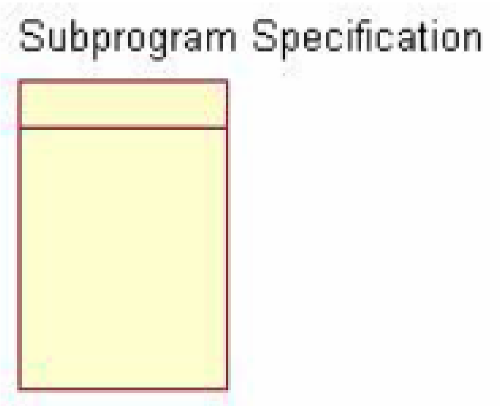
Component Diagram (cont.)

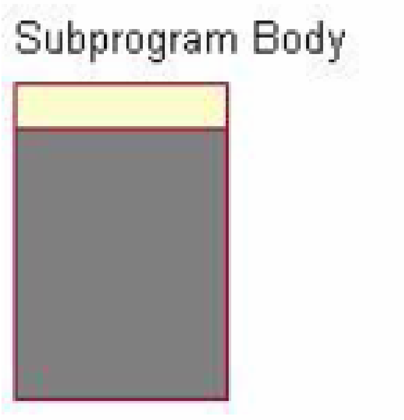
- **Component:** It represents a software module with a well-defined interface.
  - A component is a physical module of code.
  - Components can include both source code libraries and runtime files. For example, if you are using C++, each of your .cpp and .h files is a separate component.
  - In the component specification, you specify the type of component in the Stereotype field (e.g., ActiveX, Applet, Application, DLL, and Executables).



Component Diagram (cont.)

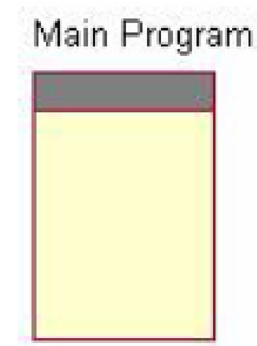
- **Subprogram Specification and Body**: They represent a subprogram's visible specification and the implementation body.
  - A subprogram is typically a collection of subroutines.





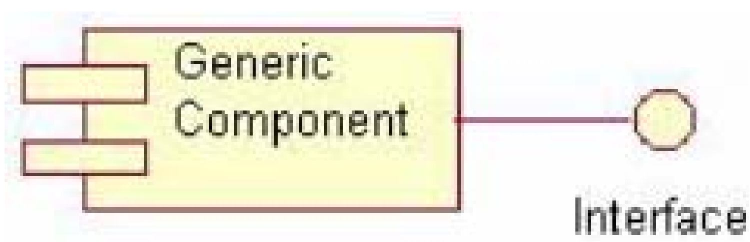
Component Diagram (cont.)

- **Main Program**: It represents the main program.
  - A main program is the file that contains the root of a program.



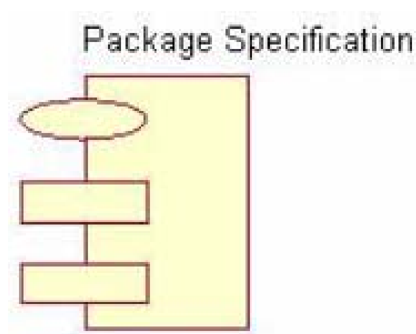
- **Database**: It represents a database, which may contain one or more schemas.

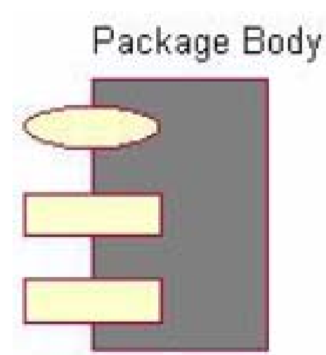
- In addition to modeling the component itself, you can model the relationship between a component and its interface.



Component Diagram (cont.)

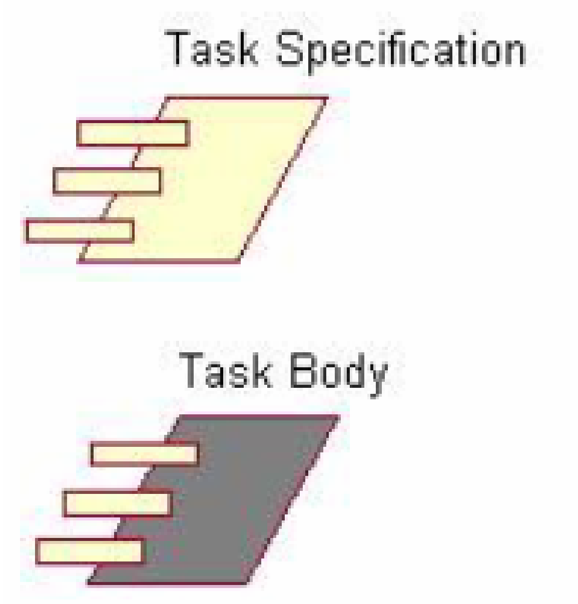
- **Package Specification and Body**: A package is the implementation of a class.
  - A package specification is a header file, which contains function prototype information for the class.
  - In C++, package specifications are the .h files. In Java, you use the Package Specification icon to represent the .java files.
  - A package body contains the code for the operations of the class. In C++, package bodies are the .cpp files.



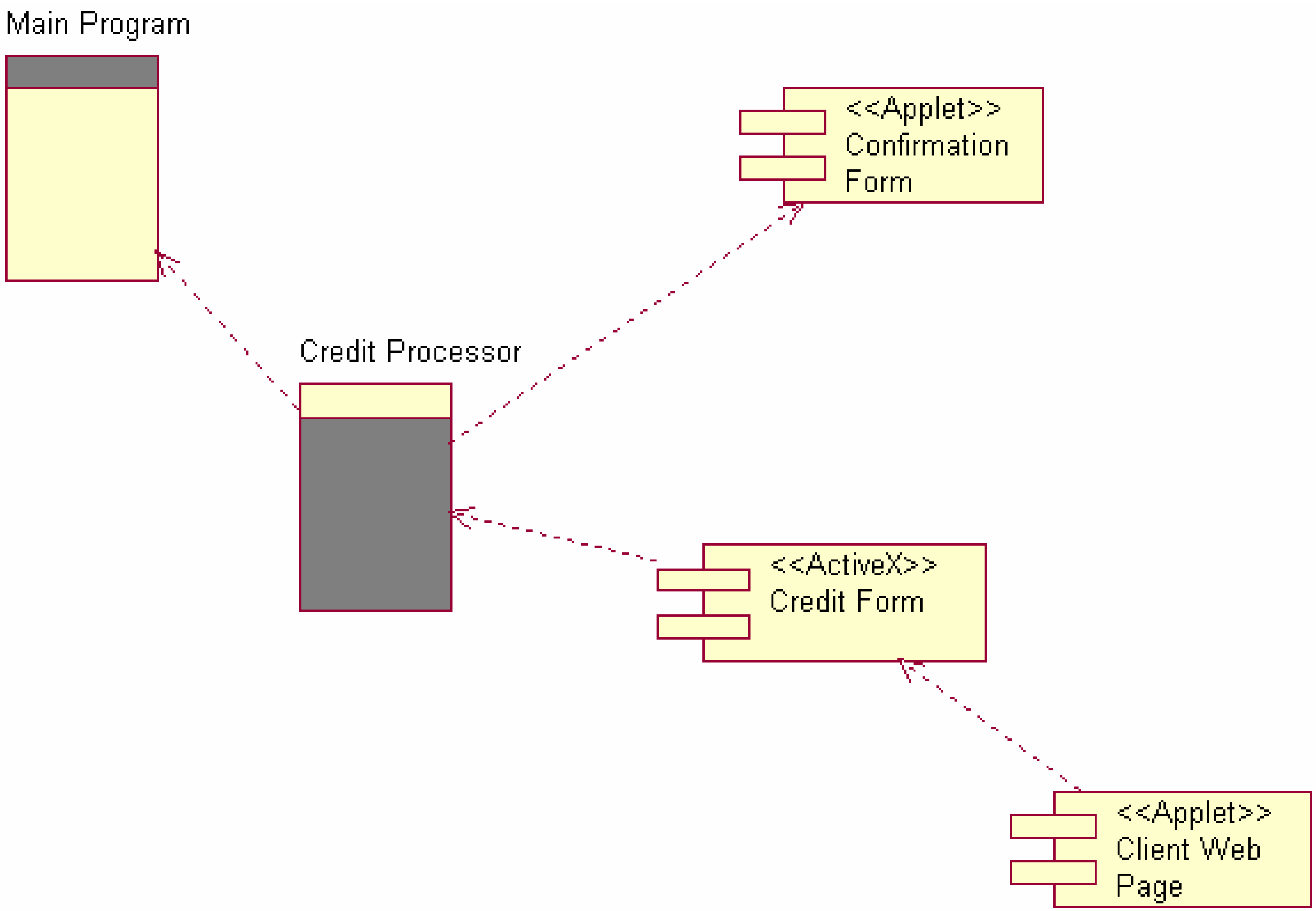


Component Diagram (cont.)

- **Task Specification and Body**: These represent packages that have independent threads of control.
  - An executable file is commonly represented as a task specification with a .exe extension.



Component Diagram (cont.)



Deployment view

- The Deployment view is concerned with the physical deployment of the system, which may differ from the logical architecture of the system.
- The Deployment view contains processors, devices, processes, and connections between processors and devices.All of this information is diagrammed on a Deployment diagram.
- There is only one Deployment diagram per system.
- For example, the system may have a logical three-tier architecture. The interface may be separated from the business logic and the database logic.
- However, the deployment may be two-tiered. The interface may be placed on one machine, while the business and database logic are located on another machine.

Deployment view (cont.)

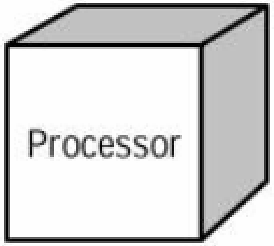
- Other issues, such as fault tolerance, network bandwidth, disaster recovery, and response time, are also handled using the Deployment view.
- Deployment view includes:
  - Processes
  - Processors
  - Connectors
  - Devices
  - Deployment diagram

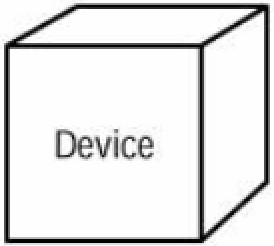
Deployment Diagram

- A Deployment diagram shows all of the nodes on the network, the connections between them, and the processes that will run on each one.
- Deployment diagrams illustrate the physical distribution of a system. A given project has one, and only one, Deployment diagram.
- These diagrams show the processors, devices, connections, and processes involved in the system.
- To open the Deployment diagram:
  - 1. Double-click Deployment View entry in the browser.
  - 2. Rose will open the Deployment diagram for the model.

Deployment Diagram (cont.)

- A processor is any machine that has processing power. The servers, workstations, and other machines with processors are included in this category.
- A device is hardware with a single purpose or a limited purpose.
  - Devices include items such as mobile terminals, printers, and scanners.
- Both processors and devices can also be referred to as nodes on a network.





Deployment Diagram (cont.)