A Tutorial on How to Create UML Diagrams

Department of Computer Science
Virginia Tech
Blacksburg, VA 24061

Version: October 8, 2009

This tutorial is prepared by compiling from the three sources cited below.

References

1. *IBM Rational Rose Enterprise* software product. (The best place to get help on drawing UML diagrams is *Rational Rose* itself. Just go to the help menu and type out the item you are searching for.)

2. http://odl-skopje.etf.ukim.edu.mk/uml-help/ (This is a very good Web site to learn more about UML)

3. UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition, http://proquest.safaribooksonline.com/0321193687/app01lev1sec2#snippet (This gives useful information about UML and its techniques.)

Table of Contents

1. Deployment Diagrams

1.1 Overview

A *deployment diagram* shows processors, devices, and connections. Each model contains a single deployment diagram that shows the connections between its processors and devices, and the allocation of its processes to processors.

Processor Specifications, Device Specifications, and Connection Specifications enable one to display and modify the respective properties. The information in a specification is presented textually; some of this information can also be displayed inside the icons. One can change properties or relationships by editing the specification or modifying the icon on the diagram. The deployment diagram specifications are automatically updated.

The deployment diagram contains nodes and connections. A node usually represents a piece of hardware in the system. A connection depicts the communication path used by the hardware to communicate and usually indicates a method such as TCP/IP.

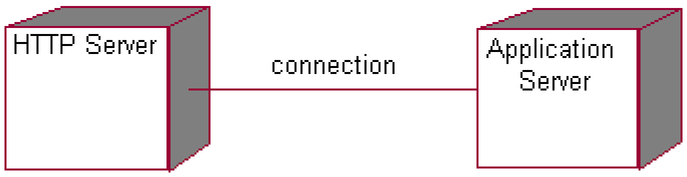


Figure 1. A deployment diagram.

1.2 Tools

1.2.1 Processor

A *processor* is a hardware component capable of executing programs. Each processor must have a name. There are no constraints on the processor name because processors denote hardware rather than software entities.

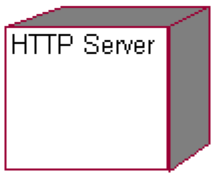


Figure 2. A deployment diagram processor.

You can create a deployment diagram in the “Deployment View” of your Rose model. Use the appropriate toolbar items by checking their labels and drag them onto the diagram window. Label the objects by right-clicking and choosing “Open Specification” and entering a label in the “Name” field.

1.2.2 Device

A *device* is a hardware component with no computing power. Each device must have a name. Device names can be generic, such as "modem" or "terminal."

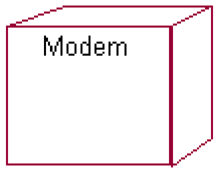


Figure 3. A deployment diagram device.

1.2.3 Connection

A *connection* represents some type of hardware coupling between two entities. An entity is either a processor or a device. The hardware coupling can be direct, such as an RS232 cable, or indirect, such as satellite-to-ground communication. Connections are usually bi-directional.

Use the “Connection” toolbar item to create connections between deployment diagram objects when drawing.

2. Component Diagrams

2.1 Overview

Component diagrams provide a physical view of the current model. A component diagram shows the organizations and dependencies among software components, including source code components, binary code components, and executable components.

These diagrams also show the externally visible behavior of the components by displaying the interfaces of the components. Calling dependencies among components are shown as dependency relationships between components and interfaces on other components. The interfaces actually belong to the logical view, but they can occur both in class diagrams and in component diagrams.

The component diagram contains *components* and *dependencies*. Components represent the physical packaging of a module of code. The dependencies between the components show how changes made to one component may affect the other components in the system. Dependencies in a component diagram are represented by a dashed line between two or more components. Component diagrams can also show the interfaces used by the components to communicate to each other.

The combined deployment and component diagram below gives a high level physical description of the completed system. The diagram shows two nodes which represent two machines communicating through TCP/IP. Component2 is dependant on component1, so changes to component 2 could affect component1. The diagram also depicts component3 interfacing with component1. This diagram gives the reader a quick overall view of the entire system.

A component diagram uses a dependency to show where a component depends on another component. In the following diagram, Component 2 depends on Component 1. The diagram also contains a database. Interfaces of components are represented by the solid line.

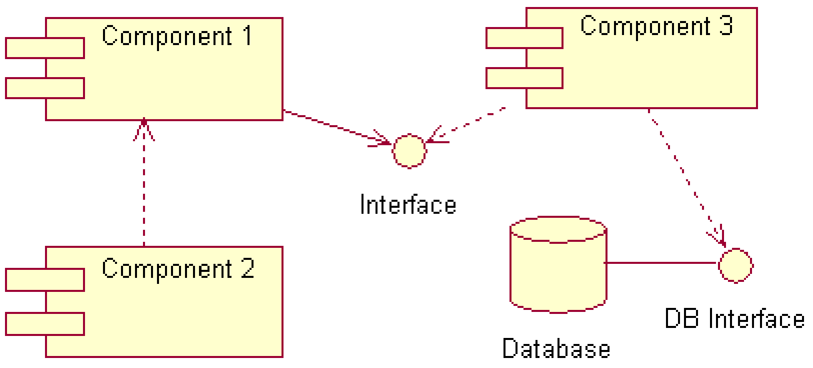


Figure 4. A component diagram.

2.2 Tools

2.2.1 Component

A *component* represents a software module (source code, binary code, executable, DLL, etc.) with a well-defined interface. The interface of a component is represented by one or several interface elements that the component provides. Components are used to show compiler and run-time dependencies, as well as interface and calling dependencies among software modules. They also show which components implement a specific class. A system may be composed of several software modules of different kinds. Each software module is represented by a component in the model. To distinguish different kinds of components from each other, stereotypes are used.

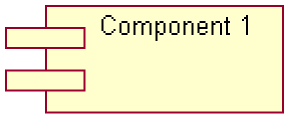


Figure 5. A component object.

To create a component diagram in Rose, right-click on Component View on the tree view on the right. Go to "New" and the choose "Component Diagram". Use the appropriate toolbar items by checking their labels.

2.2.2 Interface

An *interface* specifies the externally visible operations of a class and/or component, and has no implementation of its own. An interface typically specifies only a limited part of the behavior of a class or component.

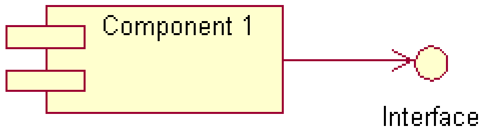


Figure 6. A component interface object.

The interface object can only be found in the "Logical View". Create the interface object by right-clicking on "Logical View" and going to "New" and then "Interface". Once it is created, you can drag it onto your component diagram. To link an interface to a component or database, right-click on the component or database object and choose "Open Specification". In the "Realizes" tab, right-click on the appropriate interface and select "Assign".

2.2.3 Database

A *database* element can also be used in the diagram. The database can be used just like a component and be connected to interface objects too. The intention is to show how the components interact with the database.

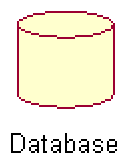


Figure 7. A database object in component diagrams.

You will have to customize the menu to add the database item. Right-click on the toolbar and select customize. A database object can also have interfaces.

2.2.4 Dependency Relationship

A *dependency* is a relationship between two model elements in which a change to one model element will affect the other model element. Use a dependency relationship to connect model elements with the same level of meaning. Typically, on class diagrams, a dependency relationship indicates that the operations of the client invoke operations of the supplier.

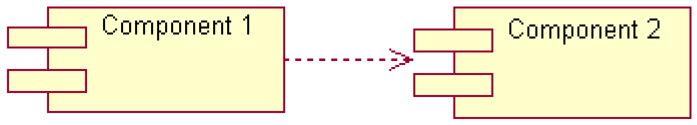


Figure 8. A dependency relationship in component diagrams.

You can connect model elements with dependencies on any diagram except state machine diagrams and object diagrams. For example, you can connect a use case to another use case, a package to another package, and a class to a package. Dependencies are also used on component diagrams to connect model elements.

3. Physical Diagrams

3.1 Overview

There are two types of *physical diagrams*: *deployment diagrams* and *component diagrams*. Deployment diagrams show the physical relationship between hardware and software in a system. Component diagrams show the software components of a system and how they are related to each other. As previously mentioned, these relationships are called dependencies.

Physical diagrams are used when development of the system is complete to describe the physical information of a system. In many cases the deployment and component diagrams are combined into one physical diagram, combining the features of both diagrams into one.

Physical diagrams can not be created in Rose. A combination of deployment and component diagrams is suggested.

4. Class Diagrams

4.1 Overview

A *class diagram* is a picture for describing generic descriptions of possible systems. They contain objects representing classes, interfaces, and their relationships. One can create a class diagram to depict the classes at the top level of a model. One can also create one or more class diagrams to depict classes contained by each package (see Section 5) in one's model. These provide an expanded view of the classes that are contained in a package.

Class diagrams are used in nearly all object oriented software designs. You can use them to describe the classes of a system and their relationships to each other.

During:	*Use-Class Diagrams To:*
Analysis	Show common roles and responsibilities of the entities that provide the system's behavior.
Design	Capture the structure of the classes that form the system's architecture.

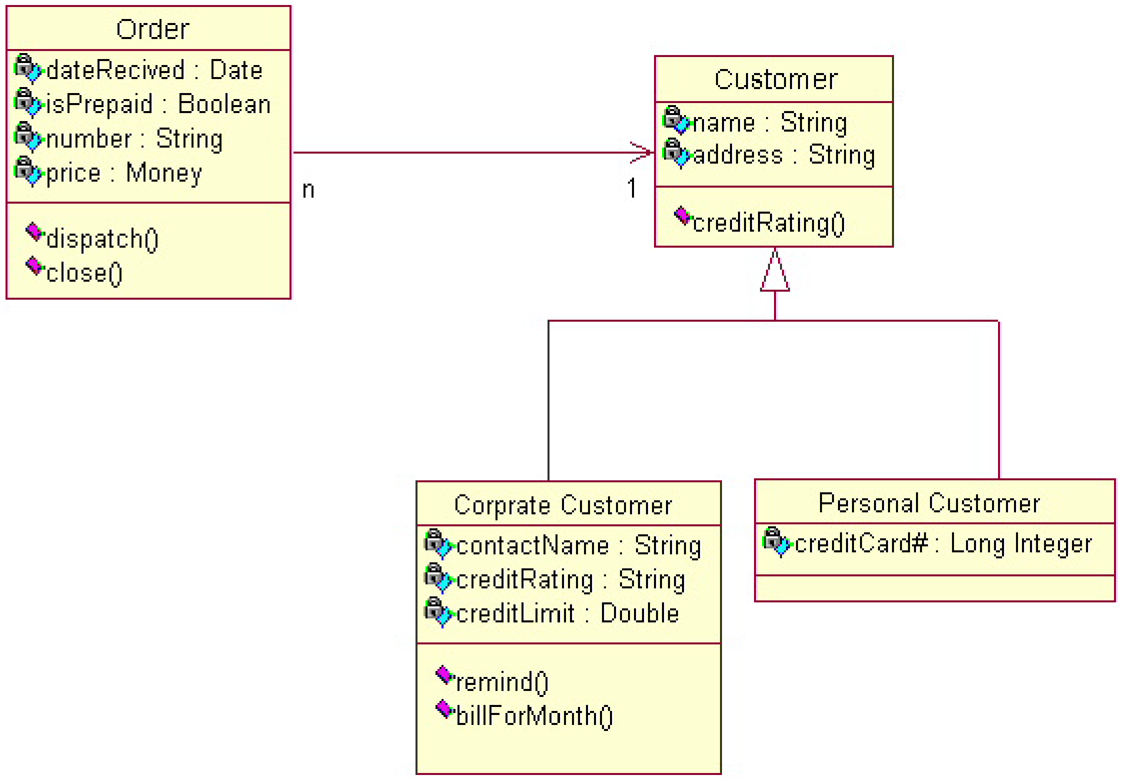


Figure 9. An example of a class diagram.

4.2 Tools

4.2.1 Classes

Class diagrams are widely used to describe the types of objects in a system and their relationships. Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagrams describe three different perspectives when designing a system, conceptual, specification, and implementation. These perspectives become evident as the diagram is created and help solidify the design. *Classes* are composed of three items: *a name*, *attributes*, and *operations*.

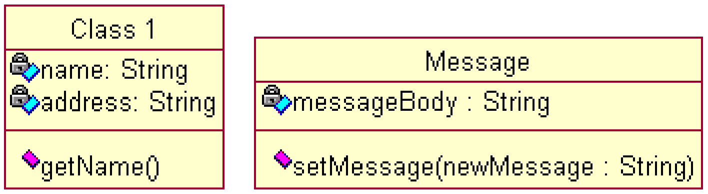


Figure 10. Examples of class objects.

Create your class diagram in the "Logical View" by right-clicking and selecting "New" and then "Class Diagram". You can create a new class by also right-click on Logical View and going to "New" and then selecting "Class". Once the class is named, drag the object onto the class diagram. Open the class specification by right-clicking on the object and choosing "Open Specification". You can add the attributes and operations by going to the corresponding tabs and right-clicking to "Insert". When creating an attribute, open the specification of the attribute you insert to set the type of the attribute. You can type it into the "Type" text area. To add arguments/parameters to an operation, open the specification of the operation once it is inserted. Go to the "Detail" tab. In the arguments section, right-click and "Insert" a new argument. Set the type for this argument by opening its own specification. Once you are done adding the attributes, go back to the diagram and right-click on the class object. Go to "Options" and then select "Show Operation Signature". This will display the attributes within your operations.

4.2.2 Interface

An *interface* specifies the externally visible operations of a class and/or component, and has no implementation of its own. An interface typically specifies only a limited part of the behavior of a class or component. Interfaces belong to the logical view but can occur in class, use case and component diagrams.

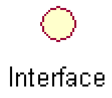


Figure 11. An interface object.

Create the interface object by right-clicking on "Logical View" in Rose and going to "New" and then "Interface".

4.2.3 Relationships and Multiplicity

Class diagrams also display *relationships* such as containment, inheritance, associations and dependencies. The *association* relationship is the most common relationship in a class diagram, showing the relationships between instances of classes. For example, in Figure 12 the class Order is associated with the class Customer. The *multiplicity* of the association denotes the number of objects that can participate in then relationship. An Order object can be associated to only one customer, but a customer can be associated to many orders.

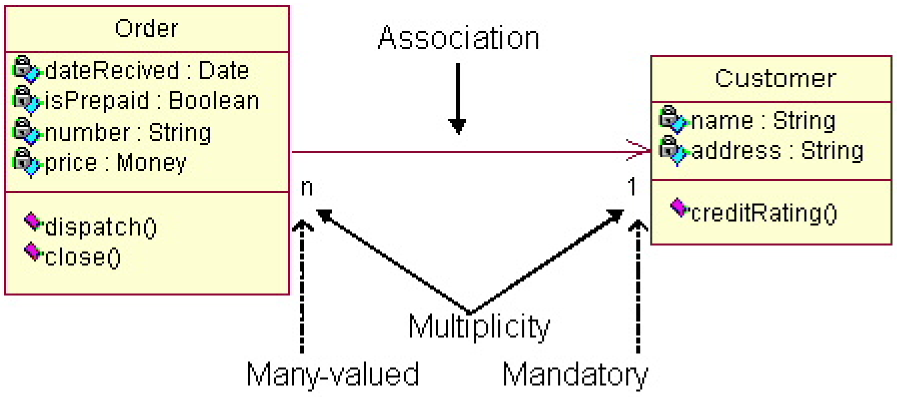


Figure 12. Relationships and multiplicity in class diagrams.

Use the appropriate toolbar items to establish the relationships. When establishing multiplicity, right-click on the appropriate relationship and "Open Specification". In the "Role A Detail" and "Role B Detail" select the multiplicity.

A *dependency* is a relationship between two model elements in which a change to one model element will affect the other model element. Use a dependency relationship to connect model elements with the same level of meaning. Typically, on class diagrams, a dependency relationship indicates that the operations of the client invoke operations of the supplier.

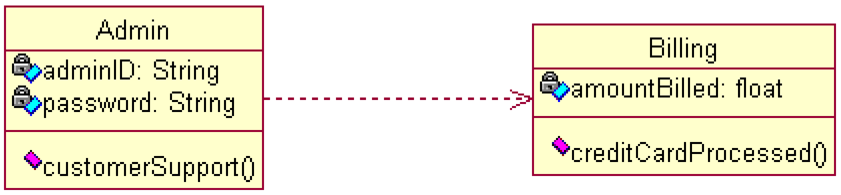


Figure 13. A dependency relationship between classes.

When an attribute of a class needs to contain a lot of information, it may just be more appropriate to make the attribute a class. Hence, a class can also contain another class if needed. The relationship between these classes in this case is called *aggregation*.

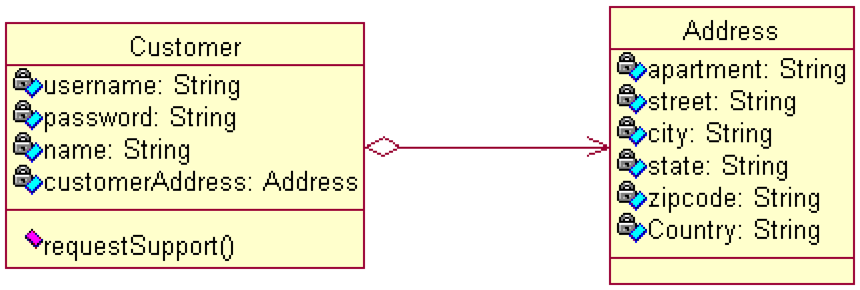


Figure 14. An aggregation relationship.

Another common relationship in class diagrams is a *generalization*, representing the concept of inheritance. A generalization is used when a class is derived from a more general class.

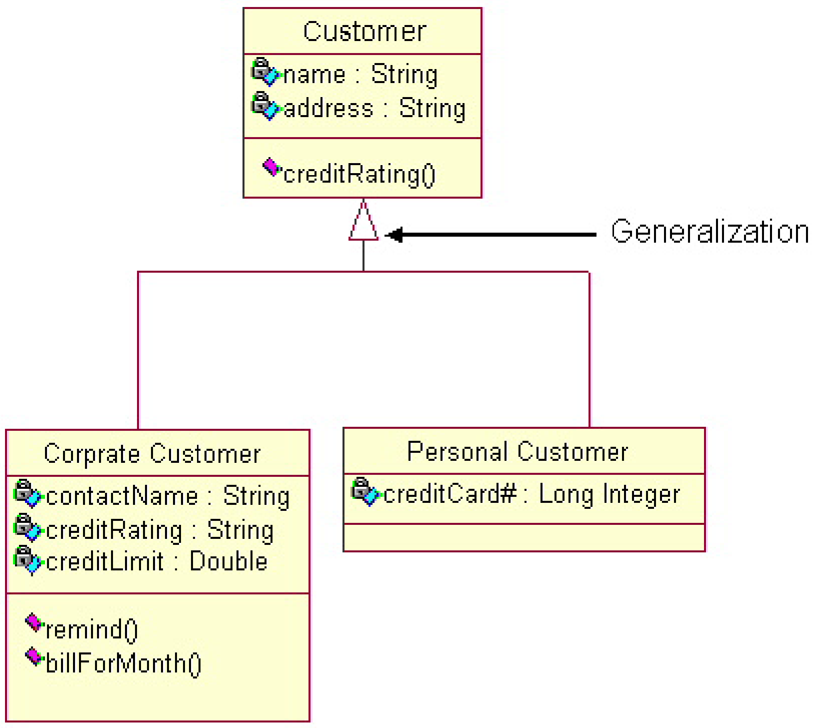


Figure 15. Generalization in class diagrams.

In Figure 15 the Corporate Customer and Personal Customer classes have some similarities such as name and address, but each class has some of its own attributes and operations. The class Customer is a general form of both the Corporate Customer and Personal Customer classes. The Corporate Customer and Personal Customer classes can inherit common characteristics from the Customer class, but can further specify their own characteristics.

A *realization* is a relationship between classes, interfaces, components, and packages that connects a client element with a supplier element. A realization relationship between classes and interfaces and between components and interfaces shows that the class realizes the operations offered by the interface.

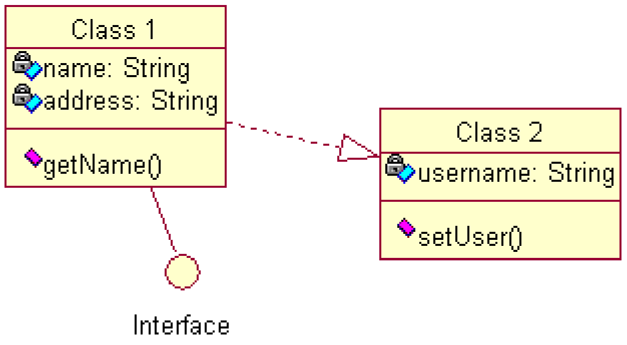


Figure 16. The realize relationship for interfaces and other objects.

When you draw a realize relationship to an interface from another model element, the realize relationship appears as a solid line. In other cases it should appear as a dashed line with an open arrowhead.

4.2.4 How to Start Class Diagrams

Class diagrams are some of the most difficult UML diagrams to create. To draw detailed and useful diagrams a person has to gain an understanding of UML and Object Oriented principles.

Before drawing a class diagram consider the three different perspectives of the system the diagram will present: conceptual, specification, and implementation. Do not focus on just one perspective, but try to understand how all work together.

When designing classes consider what attributes and operations it will have. Then try to determine how instances of the classes will interact with each other. These are the very first steps in developing a class diagram. Using these basic techniques, one can develop a complete view of a software system.

5. Package Diagrams

5.1 Overview

Package diagrams are a subset of class diagrams. They organize elements of a system into related groups to minimize the amount of dependencies. One can think of package diagrams as a higher-level view of a system when compared to class diagrams.

Such diagrams are relatively simply structured. They contain packages that are linked to other packages using dependency relationships. Packages can contain other packages in the diagrams, showing another degree of organization in the system.

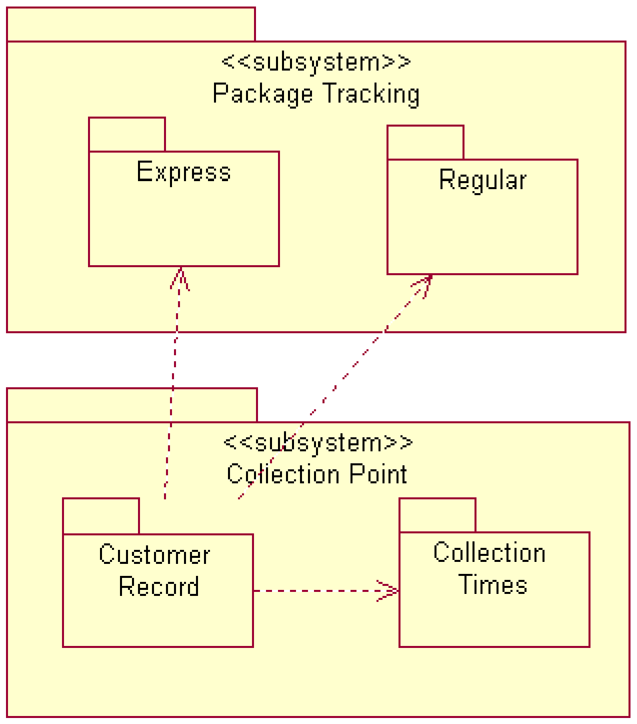


Figure 17. A package diagram.

5.2 Tools

5.2.1 Package

Component *packages* represent clusters of logically related components, or major pieces of your system. Component packages parallel the role played by logical packages for class diagrams. They allow you to partition the physical model of the system.

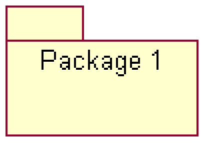


Figure 18. A component package object.

A package diagram type dose not exist in Rose. Instead, you will have to create a package diagram in another diagram such as a “Use Case Diagram”, “Class Diagram”, or “Component Diagram”. Create one of these diagrams by right-clicking on the appropriate view and going to “New” and then selecting the diagram. To create a package, right-click on the view you used to create your diagram. Go to “New” and then “Package”. You can then name and drag the package onto your diagram. You can not drag a package created in a view that does not contain the diagram you created. To show that a package contains other packages, simply place the smaller packages on top of a larger package. To show the “<<subsystem>>” label in your package, right-click on the package and open the specification and select it in the stereotype section.

5.2.2 Dependencies

A *dependency* shows that a package relies on another package for certain activities. Any changes made to one package will force the other package to also take note of the changes.

Use the “Dependency” toolbar item in your diagram to create the links between your packages when creating your diagram.

6. Interaction Diagrams

6.1 Overview

Interaction diagrams model the behavior of use cases by describing the way groups of objects interact to complete the task. The two kinds of interaction diagrams are *sequence* and *collaboration* diagrams.

6.1.1 When to Use Interaction Diagrams

Interaction diagrams are used when you want to model the behavior of several objects in a use case. They demonstrate how the objects collaborate for the behavior. Interaction diagrams do not give an in depth representation of the behavior. If you want to see what a specific object is doing for several use cases use a state diagram. To see a particular behavior over many use cases or threads use an activity diagram.

6.1.2 How to Draw Interaction Diagrams

Sequence diagrams, collaboration diagrams, or both diagrams can be used to demonstrate the interaction of objects in a use case. Sequence diagrams generally show the sequence of events that occur. Collaboration diagrams demonstrate how objects are statically connected. Both diagrams are relatively simple to draw and contain similar elements.

7. Sequence Diagrams

7.1 Overview

A *sequence diagram* is a graphical view of a scenario that shows object interaction in a time-based sequence. Sequence diagrams establish the roles of objects and help provide essential information to determine class responsibilities and interfaces.

This type of diagram is best used during early analysis phases in design because they are simple and easy to comprehend. Sequence diagrams are normally associated with use cases.

Sequence diagrams are closely related to collaboration diagrams and both are alternate representations of an interaction. There are two main differences between sequence and collaboration diagrams: sequence diagrams show time-based object interaction while collaboration diagrams show how objects associate with each other. A sequence diagram has two dimensions. The vertical axis represents elapsed time and the horizontal axis represents different objects.

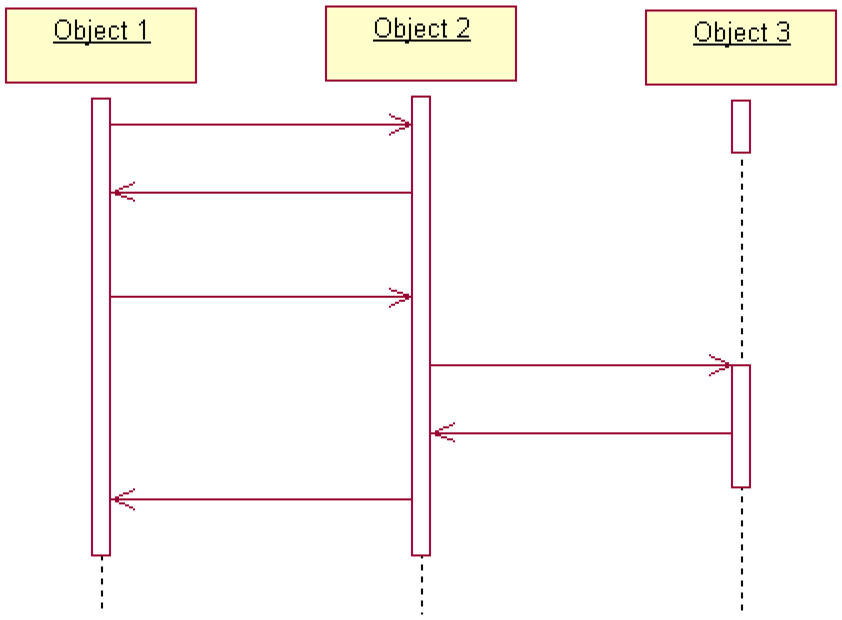


Figure 19. A sequence diagram.

Sequence diagrams demonstrate the behavior of objects in a use case by describing the objects and the messages they pass. Figure 19 shows object 1 sending a message to object 2 and receiving a reply. Object 1 sends another message to object 2, resulting in the creation of object 3 and object 2 sending and receiving messages to and from object 3. Object 3 is then destroyed and the interaction in completed when object 1 receives the final message from object 2.

7.2 Tools

7.2.1 Object

An *object* has a state, behavior and identity. The structure and behavior of similar objects are defined in their common class. Each object in a diagram indicates an instance of a class. An object that is not named is referred to as a class instance. If you use the same name for several object icons appearing in the same collaboration or activity diagram, they are assumed to represent the same object; otherwise, each object represents a distinct object. Objects appearing in different diagrams denote different objects, even if their names are identical. They can be named three different ways: object name, object name and class, or just by the class name itself.

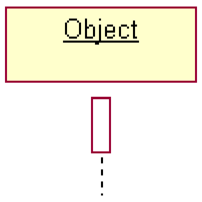


Figure 20. A sequence diagram object.

You can create a sequence diagram by right-clicking on "Logical View" or "Use Case View", going to "New" and then selecting "Sequence Diagram".

7.2.2 Messages

A *message* represents the communication between objects indicating that an action will follow. The message is shown as a horizontal solid arrow connecting two lifelines together. They can either have no label, a sequence number label, or a sequence number label with a message label. Each message represents a message passed between two objects and indicates the direction of message. While a message in a collaboration diagram can represent multiple messages, in a sequence diagram it represents exactly one message.

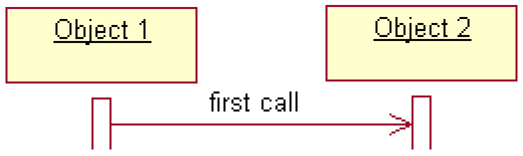


Figure 21. A message in a sequence diagram.

Use the "Object Message" toolbar item to create a message. Double-click on a message in the sequence diagram to open the specification. You can label your message in the "Name" section. The default message synchronization icon is "simple". To change the synchronization type, select a type from the synchronization field in the "Detail" tab of the message specification. To toggle the numbering of the messages in the diagram, go to "Tool" and select "Options". In the "Diagram" tab check or uncheck the "Sequence Numbering" checkbox.

8. Collaboration Diagrams

8.1 Overview

A *collaboration diagram* is an interaction diagram that shows the order of messages that implement an operation or a transaction. Collaboration diagrams show objects, their links, and their messages. They can also contain simple class instances and class utility instances. Each collaboration diagram provides a view of the interactions or structural relationships that occur between objects and object-like entities in the current model. Like sequence diagrams, collaboration diagrams contain objects. You can create one or more collaboration diagram to depict interactions for each logical package in your model.

During:	*Use Collaboration Diagrams To:*
Analysis	Indicate the semantics of the primary and secondary interactions
Design	Show the semantics of mechanisms in the logical design of the system

Use collaboration diagrams as the primary vehicle to describe interactions that express your decisions about the behavior of the system.

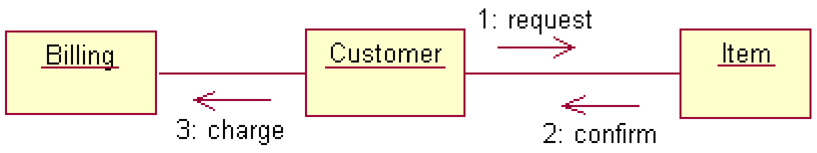


Figure 22. A collaboration diagram.

Figure 22 shows a collaboration diagram for an ordering system. A customer first requests an item. Once the request is confirmed, the customer then sends a message for billing.

8.2 Tools

8.2.1 Object

An *object* has state, behavior, and identity. The structure and behavior of similar objects are defined in their common class. Each object in a diagram indicates some instance of a class. An object that is not named is referred to as a class instance. Objects can be named in three different ways: object name, object name and class, or class name.

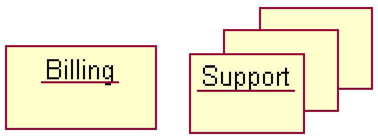


Figure 23. Objects in a collaboration diagram.

You can create the collaboration diagram in either the "Logical View" or the "Use Case View". Right-click on either of these views and go to "New" and select "Collaboration Diagram". Use the "Object" toolbar item to create an object. You can show multiple instances of an object by opening the object's specification and checking the "Multiple Instances" checkbox at the bottom.

8.2.2 Object Links and Messages

Objects interact through their links to other objects. A *link* is an instance of an association. A link should exist between two objects, including class utilities, only if there is a relationship between their corresponding classes. The existence of a relationship between two classes symbolizes a path of communication between instances of the classes: one object may send messages to another. Links can support multiple messages in either direction.

A *message* is the communication carried between two objects that trigger an event. A message carries information from the source focus of control to the destination focus of control. A message is represented on collaboration diagrams and sequence diagrams by a message icon which visually indicates its synchronization.

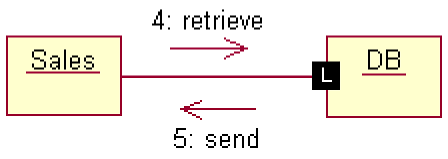


Figure 24. An object link and link messages.

Use the "Object Link", "Link Message", and "Reverse Link Message" toolbar items to link and create messages between objects. Label your messages by opening the specification of the message. To set the supplier visibility of an object link (shown by the L in the link), open the specification of the link and select appropriate one. In the above figure's case, choose "Local".

8.2.3 Asynchronous

In a collaboration diagram, a message can represent several messages. If all messages represented by a message do not have the same synchronization, a "simple" message is displayed.

The message synchronization can be changed in a message specification. Open the specification and choose the appropriate type in the "Detail" tab.

9. Activity Diagrams

9.1 Overview

Activity diagrams describe the workflow behavior of a system. The diagrams describe the state of activities by showing the sequence of activities performed. Activity diagrams can show activities that are conditional or parallel, providing a way to model the workflow of a process similar to a flowchart. An activity diagram can also be used to model code-specific information such as a class operation.

The diagram is a special case of a state machine in which most of the states are activities and most of the transitions are implicitly triggered by completion of the actions in the source activities. The main difference between activity diagrams and statecharts is the activity-centric nature as opposed to the state-centric nature of state charts. An activity diagram is typically used for modeling the sequence of activities in a process, whereas a statechart is better suited to model the discrete stages of an object’s lifetime.

9.1.1 When to Use Activity Diagrams

Activity diagrams should be used in conjunction with other modeling techniques such as interaction diagrams and state diagrams. Activity Diagrams are also useful for analyzing a use case by describing what actions need to take place and when they should occur, describing a complicated sequential algorithm, and modeling applications with parallel processes. Activity diagrams should not take the place of interaction diagrams and state diagrams as they do not give any details on how objects should interact.

9.1.2 How to Draw Activity Diagrams

Diagrams are read from top to bottom and have forks to describe conditions and parallel activities. A fork is used when multiple activities are occurring at the same time. Figure 25 shows an activity diagram that starts with a start state. Two activities are done sequentially and then the browsing activities can occur in parallel. This is marked by the horizontal bar. The activities are then synchronized, leading to three more sequential activities. The diagram ends with an end state.

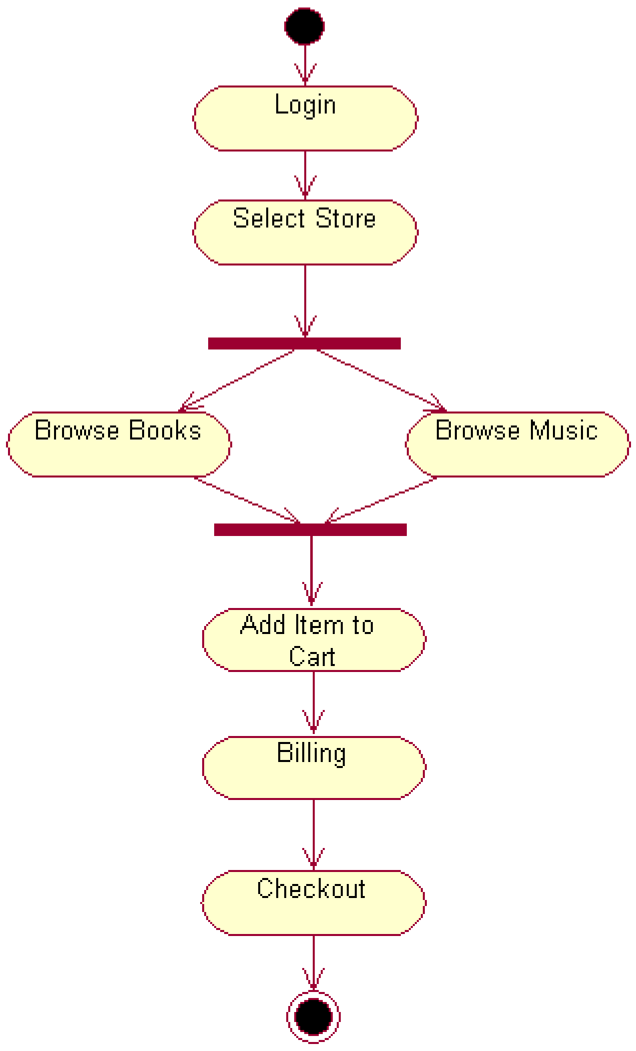


Figure 25. An activity diagram.

9.2 Tools

9.2.1 Activities

An *activity* represents the performance of task or duty in a workflow. It may also represent the execution of a statement in a procedure. An activity is similar to a state, but expresses the intent that there is no significant waiting (for events) in an activity. Transitions connect activities with other model elements and object flows connect activities with objects.

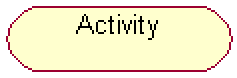


Figure 26. An activity diagram activity object.

You can create an activity diagram in the “Logical View” or “Use Case View”. Right-click on the view and go to “New” and select “Activity Diagram”. The new diagram will be created under a “State/Activity Model” category in the view.

The name of an activity must be unique and should declare the activity’s purpose. All activity icons with the same name on an activity diagram represent the same activity.

9.2.2 Transitions

A *state transition* indicates that an object in the source state will perform certain specified actions and enter the destination state when a specified event occurs or when certain conditions are satisfied. A state transition is a relationship between two activities. You can show one or more state transitions from a state as long as each transition is unique. Transitions originating from a state cannot have the same event, unless there are conditions on the event.

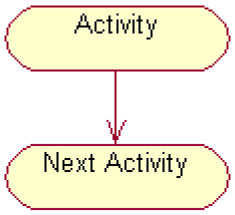


Figure 27. A state transition shown by the solid arrow.

9.2.3 Start and End States

A *start state* explicitly shows the beginning of a workflow on an activity diagram. You can have only one start state for each state machine because each workflow/execution of a state machine begins in the same place.

An *end state* represents a final or terminal state on an activity diagram or statechart diagram. Place an end state when you want to explicitly show the end of a workflow on an activity diagram or the end of a statechart diagram. Transitions can only occur into an end state; however, there can be any number of end states per context.

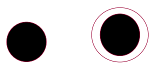


Figure 28. A start and end state object.

The same start state object can be placed on multiple diagrams if needed when working with multiple activity or statechart diagrams.

9.2.4 Decisions

A *decision* represents a specific location on an activity diagram or statechart diagram where the workflow may branch based upon guard conditions. There may be more than two outgoing transitions with different guard conditions, but for the most part, a decision will have only two outgoing transitions determined by a Boolean expression. Refer to Section 10 for more on the decision object.

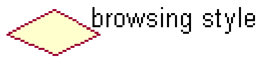


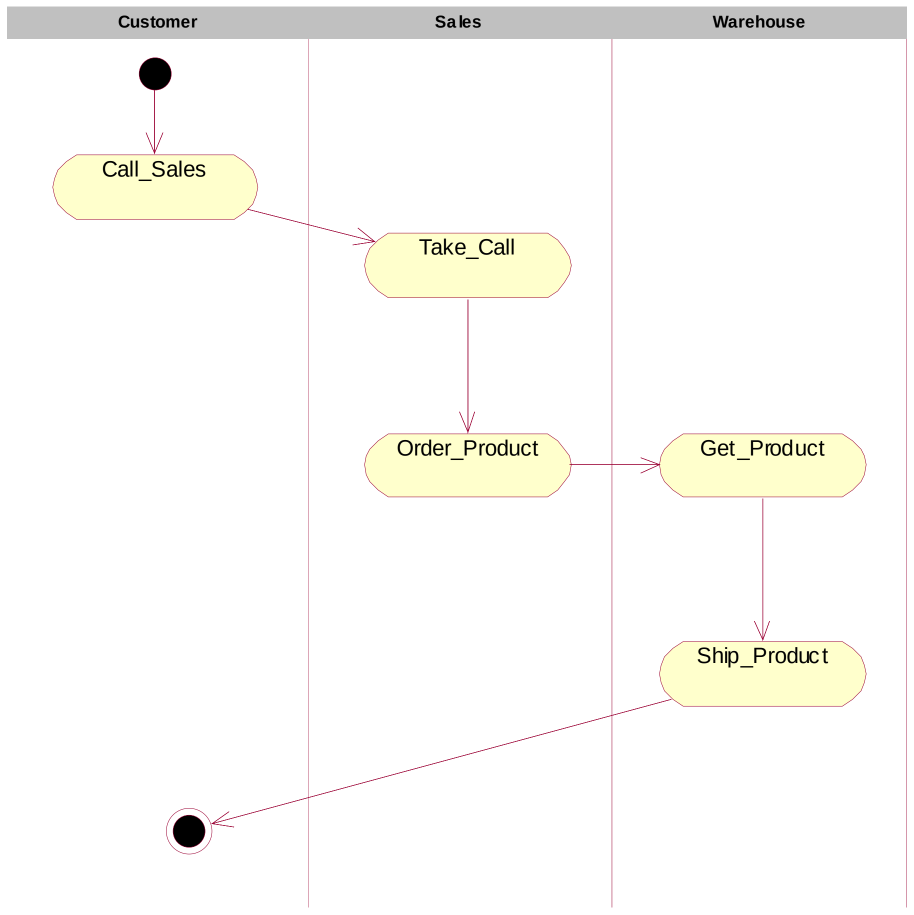
Figure 29. An activity diagram decision object.

9.2.5 Swimlanes

Swimlanes are helpful when modeling a business workflow because they can represent organizational units or roles within a business model. Swimlanes are very similar to an object because they provide a way to tell who is performing a certain role. Swimlanes only appear on activity diagrams. You should place activities within swimlanes to determine which unit is responsible for carrying out the specific activity.

When a swimlane is dragged onto an activity diagram, it becomes a swimlane view. Swimlanes appear as small icons in the browser while a swimlane views appear between the thin, vertical lines with a header that can be renamed and relocated.

Graphical Depiction



9.2.6 Synchronization

Synchronizations enable you to see a simultaneous workflow in an activity diagram or statechart diagram. Synchronizations visually define forks and joins representing parallel workflow.

A fork construct is used to model a single flow of control that divides into two or more separate, but simultaneous flows. Every fork that appears on an activity diagram should ideally be accompanied by a corresponding join. A join consists of two of more flows of control that unite into a single flow of control. All model elements that appear between forks and joins must complete before the flow of controls can unite into one.

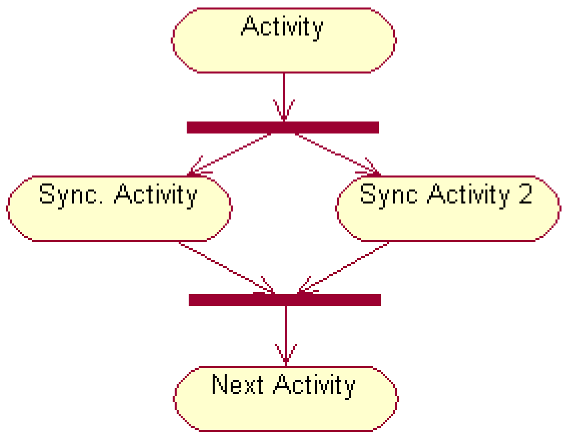


Figure 30. Horizontal synchronization in activity diagrams.

The fork and joins are represented as a horizontal bar. Use the "Horizontal Synchronization" toolbar item for this aspect of the diagram.

10. State Diagrams

State diagrams or *statechart diagrams* are used to describe the behavior of a system by showing all the possible states of an object as events occur. Each diagram usually represents objects of a single class and tracks the different states of its objects through the system.

10.1 Overview

Statechart diagrams model the dynamic behavior of individual classes or any other kind of object. They show the sequences of states that an object goes through, the events that cause a transition from one state to another, and the actions that result from a state change.

Statechart diagrams are closely related to activity diagrams. The main difference between activity diagrams and statechart is the activity-centric nature as opposed to the state-centric nature of state charts. An activity diagram is typically used for modeling the sequence of activities in a process, whereas a statechart is better suited to model the discrete stages of an object's lifetime.

Each state in the diagram represents a named condition during the life of an object during which it satisfies some condition or waits for some event. A statechart diagram typically contains one start state and multiple end states. Transitions connect the various states on the diagram.

10.1.1 When to Use State Diagrams

Use state diagrams to demonstrate the behavior of an object through many use cases of the system. Only use state diagrams for classes where it is necessary to understand the behavior of the object through the entire system. Not all classes will require a state diagram and state diagrams are not useful for describing the collaboration of all objects in a use case.

10.1.2 How to Draw State Diagrams

State diagrams have very few elements. State diagrams begin with an initial state of an object. This is the state of the object when it is created. After the initial state, the object begins changing states. Conditions based on the activities can determine what the next state the object transitions to.

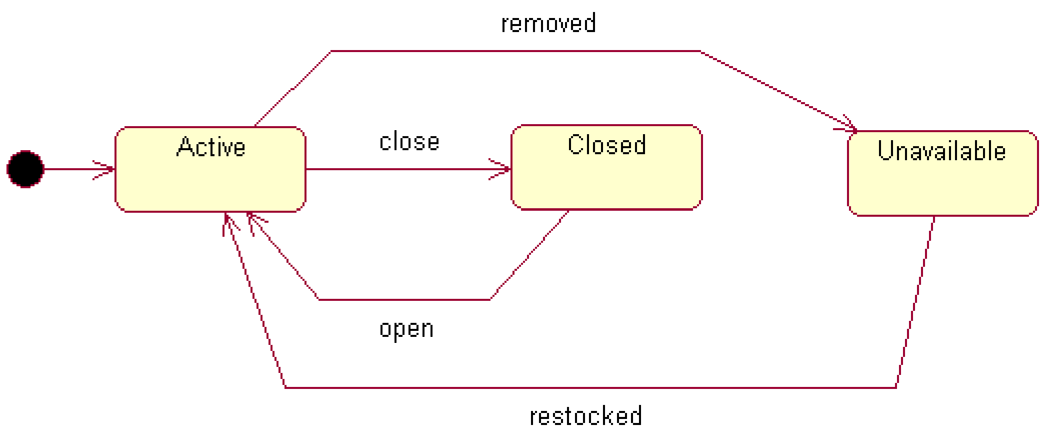


Figure 31. A statechart diagram.

In Figure 31 the diagram starts from the start state and enters the Active state. Depending on the events that occur, the state can transition to Closed or Unavailable.

10.2 Tools

10.2.1 States

A *state* represents a condition or situation during the life of an object during which it satisfies some condition or waits for some event. Each state represents the cumulative history of its behavior.

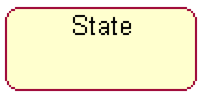


Figure 32. A state object.

To create a statechart diagram, right-click on “Logical View” or “Use Case View” and then go to “New” and select “Statechart Diagram”. The diagram will be created in the “State/Activity Diagram” category.

10.2.2 Start and End States

Refer to Section 9 for start and end states.

10.2.3 State Transition

A *state transition* indicates that an object in the source state will perform certain specified actions and enter the destination state when a specified event occurs or when certain conditions are satisfied. A state transition is a relationship between two states, two activities, or between an activity and a state. You can show one or more state transitions from a state as long as each transition is unique. Transitions originating from a state cannot have the same event, unless there are conditions on the event.

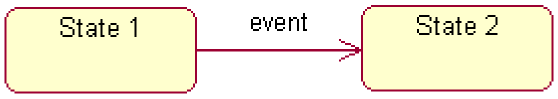


Figure 33. A transition in a statechart diagram.

10.2.4 Decision

A *decision* represents a specific location on a statechart diagram where the workflow may branch based upon guard conditions. There may be more than two outgoing transitions with different guard conditions, but for the most part, a decision will have only two outgoing transitions determined by a Boolean expression.

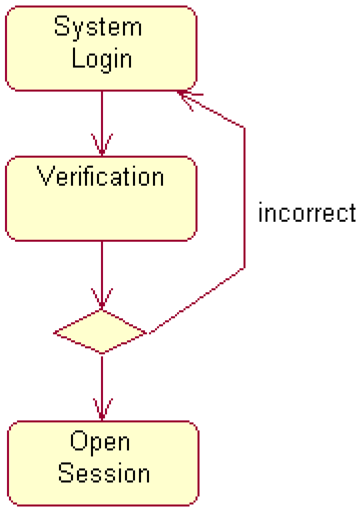


Figure 34. A decision in a statechart diagram.

The decision toolbar item will have to be added into the toolbar. Right-click on the toolbar to customize it and add the item.