UNIFIED MODELING LANGUAGE (UML)

The Unified Modeling Language (UML) is a general-purpose visual modeling language that is used to specify, visualize, construct, and document the artifacts of a software system. It captures decisions and understanding about systems that must be constructed. It is used to understand, design, browse, configure, maintain, and control information about such systems. It is intended for use with all development methods, lifecycle stages, application domains, and media. The modeling language is intended to unify past experience about modeling techniques and toincorporate current software best practices into a standard approach. UML includes semantic concepts, notation, and guidelines. It has static, dynamic, environmental, and organizational parts. It is intended to be supported by interactive visual modeling tools that have code generators and report writers. The UML captures information about the static structure and dynamic behavior of a system. A system is modeled as a collection of discrete objects that interact to perform work that ultimately benefits an outside user. The static structure defines the kinds of objects important to a system and to its implementation, as well as the relationships among the objects. The dynamic behavior defines the history of objects over time and the communications among objects to accomplish goals. Modeling a system from several separate but related viewpoints permits it to be understood for different purposes.
A model is a representation in a certain medium of something in the same or another medium. The model captures the important aspects of the thing being modeled from a certain point of view and simplifies or omits the rest. Engineering, architecture, and many other creative fields use models. A model is expressed in a medium that is convenient for working. Models of buildings may be drawings on paper, 3-D figures made of cardboard and papiermâché, or finite-element equations in a computer. A construction model of a building shows the appearance of the building but can also be used to make engineering and cost calculations. A model of a software system is made in a modeling language, such as UML. The model has both semantics and notation and can take various forms that include both pictures and text. The model is intended to be easier to use for certain purposes than the final system. Models are used:

1) To capture and precisely state requirements and domain knowledge so that all stakeholders may understand and agree on them.
2) To think about the design of a system.
3) To capture design decisions in a mutable form separate from the requirements.
4) To generate usable work products.
5) To organize, find, filter, retrieve, examine, and edit information about large systems.
6) To explore multiple solutions economically.
7) To master complex systems.

UML views

There is no sharp line between the various concepts and constructs in UML, but, for convenience, we divide them into several views. A view is simply a subset of UML modeling constructs that represents one aspect of a system. The divisioninto different views is somewhat arbitrary, but we hope it is intuitive. One or twokinds of diagrams provide a visual notation for the concepts in each view.At the top level, views can be divided into three areas: structural classification, dynamic behavior, and model management. Structural classification describes the things in the system and their relationships to other things. Classifiers include classes, use cases, components, and nodes. Classifiers provide the basis on top of which dynamic behavior is built. Classification views include the static view, use case view, and implementation view. Dynamic behavior describes the behavior of a system over time. Behavior canbe described as a series of changes to snapshots of the system drawn from the static view. Dynamic behavior views include the state machine view, activity view, and interaction view. Model management describes the organization of the models themselves in to hierarchical units. The package is the generic organizational unit for models. Special packages include models and subsystems. The model management view crosses the other views and organizes them for development work and configuration control. UML also contains several constructs intended to provide a limited but useful extensibility capability. These constructs include constraints, stereotypes, andtagged values. These constructs are applicable to elements of all views.

UML Diagrams

The example used here is a theater box office that has computerized its operations.This is a contrived example, the purpose of which is to highlight variousUML constructs in a brief space. It is deliberately simplified and is not presentedin full detail. Other sample examples are also provided.

Table : *UML Views and Diagrams*

Major Area	*View*	*Diagrams*	*Main Concepts*
structural	static view	class diagram	class, association, generalization, dependency, realization, interface
	use case view	use case diagram	use case, actor, association, extend, include, use case generalization
	implementation view	component diagram	component, interface, dependency, realization
	deployment view	deployment diagram	node, component, dependency, location
dynamic	state machine view	statechart diagram	state, event, transition, action
	activity view	activity diagram	state, activity, completion transition, fork, join
	interaction view	sequence diagram	interaction, object, message, activation
		collaboration diagram	collaboration, interaction, collaboration role, message
model management	model management view	class diagram	package, subsystem, model
extensibility	all	all	constraint, stereotype, tagged values

Use-case diagram

A use case illustrates a unit of functionality provided by the system. Themain purpose of the use-case diagram is to help development teamsvisualize the functional requirements of a system, including the relationship of "actors" (human beings who will interact with the system) to essential processes, as well as the relationships among different use cases. Use-case diagrams generally show groups of use cases -- either alluse cases for the complete system, or a breakout of a particular group ofuse cases with related functionality (e.g., all security administration related use cases). To show a use case on a use-case diagram, you draw an oval in the middle of the diagram and put the name of the use case in the center of, or below, the oval. To draw an actor (indicating a system user) on a use-case diagram, you draw a stick person to the left or right of your diagram (and just in case you're wondering, some people draw prettier stick people than others). Use simple lines to depict relationships between actors and use cases, as shown in Figure below.

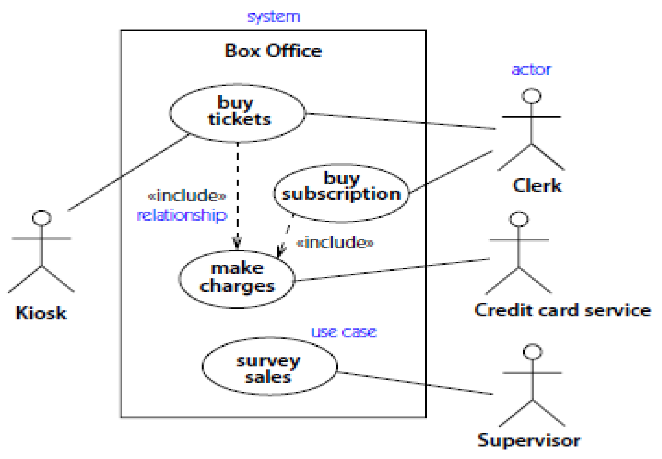

Figure *Use case diagram*

In the above Use Case, actors includethe clerk, supervisor, and kiosk. The kiosk is another system that accepts ordersfrom a customer. The customer is not an actor in the box office application because the customer is not

directly connected to the application. Use cases include buying tickets through the kiosk or the clerk, buying subscriptions (only through the clerk), and surveying total sales (at the request of the supervisor). Buying tickets and buying subscriptions include a common fragment—that is, making charges to the credit card service. (A complete description of a box office systemwould involve a number of other use cases, such as exchanging tickets and checking availability.) A use-case diagram is typically used to communicate the high-level functions of the system and the system's scope.
In addition, the absence of use cases in this diagram shows what the system doesn't do. With clear and simple use-case descriptions provided on such a diagram, a project sponsor caneasily see if needed functionality is present or not present in the system.

Class diagram

Below is the class diagram from the box office application. This diagramcontains part of a ticket-selling domain model. It shows several important classes, such as Customer, Reservation, Ticket, and Performance. Customers may have many reservations, but each reservation is made by one customer. Reservations areof two kinds: subscription series and individual reservations. Both reserve tickets: in one case, only one ticket; in the other case, several tickets. Every ticket is part of a subscription series or an individual reservation, but not both. Every performance has many tickets available, each with a unique seat number. A performance can be identified by a show, date, and time.
The class diagram shows how the different entities (people, things, and data) relate to each other; in other words, it shows the static structures of the system. A class diagram can be used to display logical classes, whichare typically the kinds of things the business people in an organization talkabout. Class diagrams can also be used to show implementation classes, which are the things that programmers typically deal with.

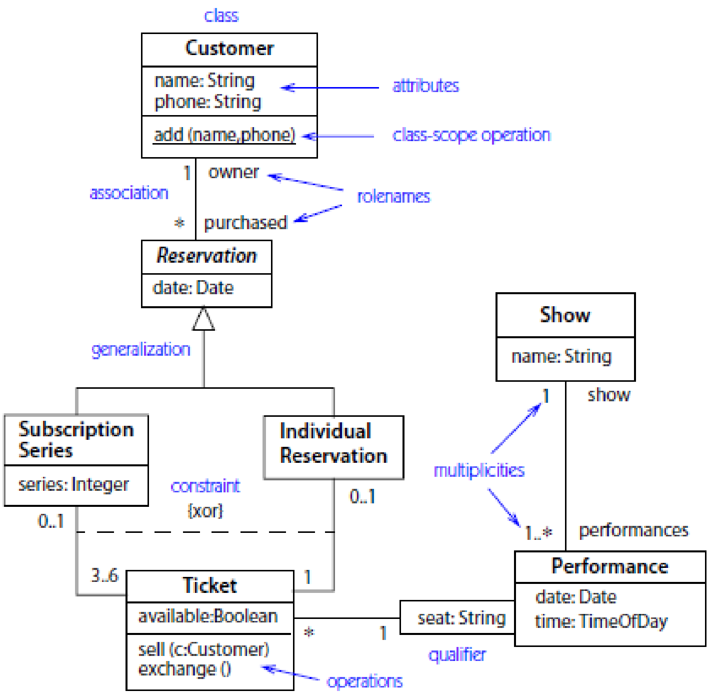

Figure . *Class diagram*

A class is depicted on the class diagram as a rectangle with three horizontal sections, as shown. The upper section shows the class's name; the middle section contains the class's attributes; and the lower section contains the class's operations (or "methods"). Relationships among classes are drawn as paths connecting class rectangles. The different kinds of relationships are distinguished by line texture and by adornments on the paths or their ends.
Draw the inheritance relationship using a line with an arrowhead at the top pointing to thesuper class, and the arrowhead should be a completed triangle. An association relationship should be a solid line if both classes are aware of each other and a line with an open arrowhead if the association is knownby only one of the classes.

Sequence diagram

Sequence diagrams **show a detailed flow for a specific use case or even just part of a specific use case**. They are almost self explanatory; they show the calls between the different objects in their sequence and can show, at a detailed level, different calls to different objects.A sequence diagram has two dimensions: The vertical dimension shows the sequence of messages/calls in the time order that they occur; thehorizontal dimension shows the object instances to which the messages are sent.

A **sequence diagram shows a set of messages arranged in time sequence**. Each classifier role is shown as a lifeline—that is, a vertical line that represents the role overtime through the entire interaction. Messages are shown as arrows between lifelines.A sequence diagram can show a scenario—that is, an individual history of atransaction.One use of a sequence diagram is to show the behavior sequence of a use case.When the behavior is implemented, each message on a sequence diagram correspondsto an operation on a class or an event trigger on a transition in a state machine.

Figure below shows a sequence diagram for the buy tickets use case. This use case is initiated by the customer at the kiosk communicating with the box office. Thesteps for the make charges use case are included within the sequence, which involves communication with both the kiosk and the credit card service.

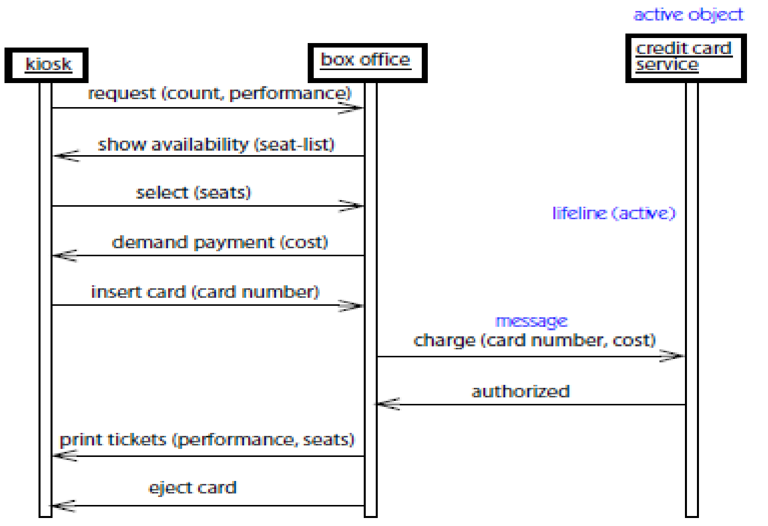
active object
kiosk
box office
credit card service
request (count, performance)
show availability (seat-list)
select (seats)
lifeline (active)
demand payment (cost)
insert card (card number)
message
charge (card number, cost)
authorized
print tickets (performance, seats)
eject card

Figure *Sequence diagram*

A sequence diagram is very simple to draw. Across the top of your diagram, identify the class instances (objects) by putting each class instance inside a box. In the box, put the class instance name and class name separated by a space/colon/space " : ". If a class instance sends amessage to another class instance, draw a line with an open arrowhead pointing to the receiving class instance; place the name of the message/method above the line. Optionally, for important messages, you can draw a dotted line with an arrowhead pointing back to the originating class instance; label the return value above the dotted line.

Collaboration diagram

A collaboration models the objects and links that are meaningful within an interaction.The objects and links are meaningful only in the context provided by theinteraction. A classifier role describes an object and an association role describes alink within a collaboration. A collaboration diagram shows the roles in the interactionas a geometric arrangement (Figure below). The messages are shown as arrowsattached to the relationship lines connecting classifier roles. The sequence ofmessages is indicated by sequence numbers prepended to message descriptions.One use of a collaboration diagram is to show the implementation of an operation.The collaboration shows the parameters and local variables of the operation,as well as more permanent associations. When the behavior is implemented, themessage sequencing corresponds to the nested calling structure and signal passingof the program.

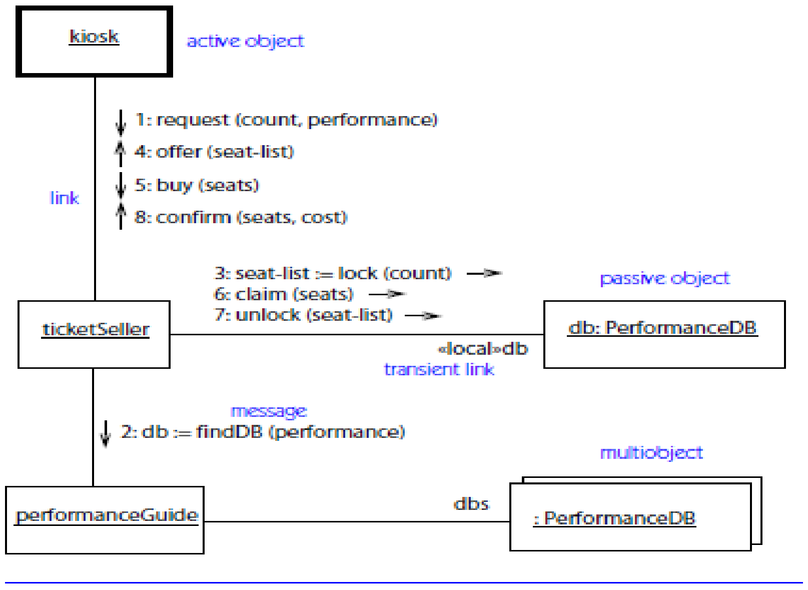

Figure *Collaboration diagram*

Figure above shows a collaboration diagram for the reserve tickets interaction at alater stage of development. The collaboration shows the interaction among internalobjects in the application to reserve tickets. The request arrives from the kioskand is used to find the database for the particular performance from the set of allperformances. The pointer db that is returned to the ticketSeller object represents a local transient link to a performance database that is maintained during the interactionand then discarded. The ticket seller requests a number of seats to theperformance; a selection of seats in various price ranges is found, temporarilylocked, and returned to the kiosk for the customer's selection. When the customermakes a selection from the list of seats, the selected seats are claimed and the restare unlocked.

Both sequence diagrams and collaboration diagrams show interactions, butthey emphasize different aspects. A sequence diagram shows time sequence as ageometric dimension, but the relationships among roles are implicit. A collaborationdiagram shows the relationships among roles geometrically and relates messagesto the relationships, but time sequences are less clear because they areimplied by the sequence numbers. Each diagram should be used when its main aspectis the focus of attention.

State chart Diagram

A state machine models the possible life histories of an object of a class. A statemachine contains states connected by transitions. Each state models a period oftime during the life of an object during which it satisfies certain conditions. Whenan event occurs, it may cause the firing of a transition that takes the object to anew state. When a transition fires, an action attached to the transition may be executed.State machines are shown as statechart diagrams. Figure below shows a statechart diagram for the history of a ticket to a performance.

The initial state of a ticket (shown by the black dot) is the Available state.Before the season starts, seats for season subscribers are assigned. Individual ticketspurchased interactively are first locked while the customer makes a selection.After that, they are either sold or unlocked if they are not chosen. If the customertakes too long to make a selection, the transaction times out and the seat is released.Seats sold to season subscribers may be exchanged for other performances,in which case they become available again.

State machines may be used to describe user interfaces, device controllers, andother reactive subsystems. They may also be used to describe passive objects thatgo through several qualitatively distinct phases during their lifetime, each of whichhas its own special behavior.

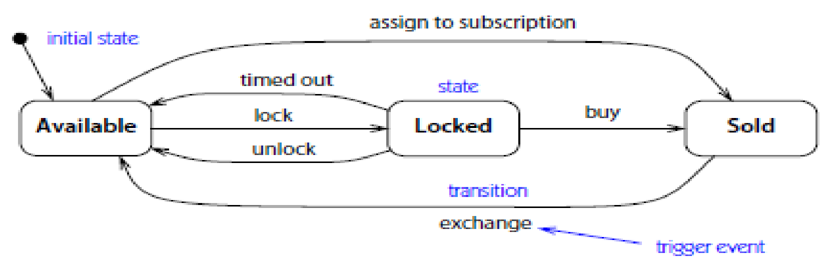

Figure *Statechart diagram*

Activity Diagram
An activity graph is a variant of a state machine that shows the computational activities involved in performing a calculation. An activity state represents an activity:a workflow step or the execution of an operation. An activity graph describesboth sequential and concurrent groups of activities. Activity graphs are shown on activity diagrams. Figure below shows an activity diagram for the box office. This diagram shows theactivities involved in mounting a show. (Don't take this example too seriously ifyou have theater experience!) Arrows show sequential dependencies—for example, shows must be picked before they are scheduled. Heavy bars show forks orjoins of control. For example, after the show is scheduled, the theater can begin to publicize it, buy scripts, hire artists, build sets, design lighting, and make costumes, all concurrently. Before rehearsal can begin, however, the scripts must be ordered and the artist must be hired.
This example shows an activity diagram the purpose of which is to model the real-world workflows of a human organization. Such business modeling is a major purpose of activity diagrams, but activity diagrams can also be used for modeling software activities. An activity diagram is helpful in understanding the high-level execution behavior of a system, without getting involved in the internal details of message passing required by a collaboration diagram. The input and output parameters of an action can be shown using flow relationshipsconnecting the action and an object flow state.

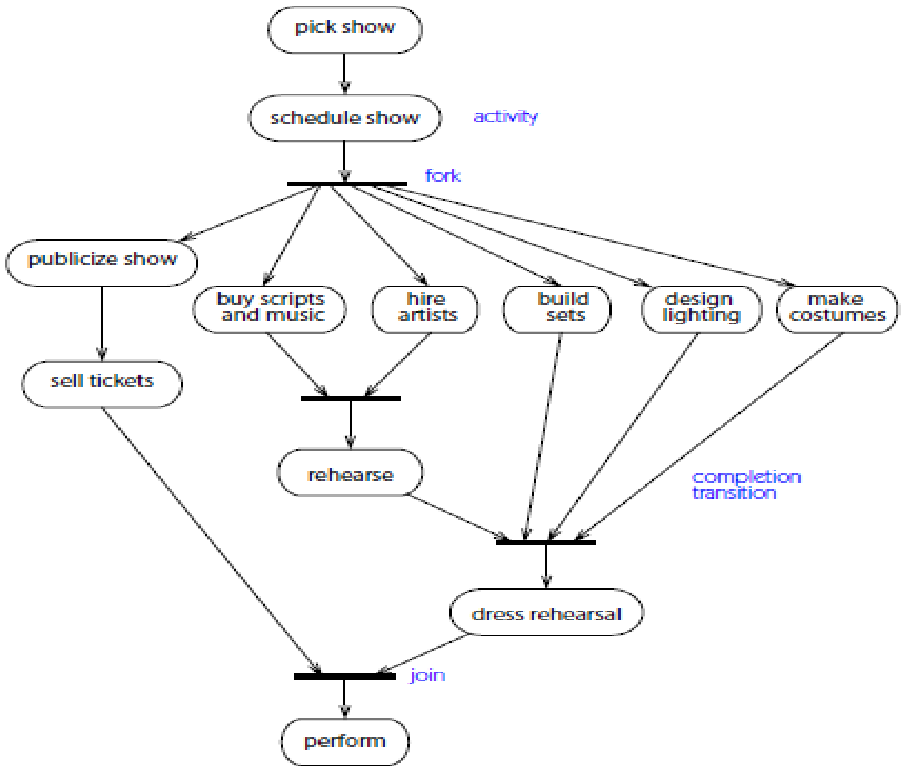

Figure *Activity diagram*

More about UML diagrams:

Component Diagram

The physical views model the implementation structure of the application itself, such as its organization into components and its deployment onto run-time nodes. These views provide an opportunity to map classes onto implementation components and nodes. There are two physical views: the implementation view and the deployment view. The implementation view models the components in a system—that is, the software units from which the application is constructed—as well as the dependencies among components so that the impact of a proposed change can be assessed. It also models the assignment of classes and other model elements to components.

The implementation view is displayed on component diagrams. Figure below shows a component diagram for the box office system. There are three user interfaces: one each for customers using a kiosk, clerks using the on-line reservation system, and supervisors making queries about ticket sales. There is a ticket seller component that sequentializes requests from both kiosks and clerks; a component that processes credit card charges; and the database containing the ticket information. The component diagram shows the kinds of components in the system; a particular configuration of the application may have more than one copy of a component.

A small circle with a name is an interface—a coherent set of services. A solid line from a component to an interface indicates that the component provides the services listed in the interface. A dashed arrow from a component to an interface indicates that the component requires the services provided by the interface. For example, subscription sales and group sales are both provided by the ticket seller component; subscription sales are accessible from both kiosks and clerks, but group sales are only accessible from a clerk.

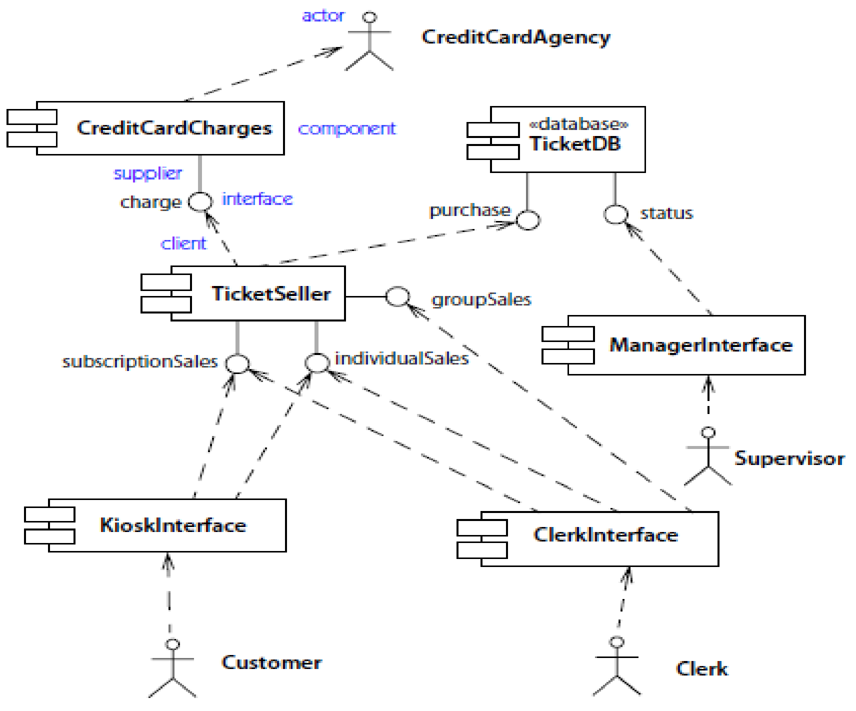

Figure 3-7. *Component diagram*

Deployment Diagram

The deployment view represents the arrangement of run-time component instances on node instances. A node is a run-time resource, such as a computer, device, or memory. This view permits the consequences of distribution and resource allocation to be assessed. The deployment view is displayed on deployment diagrams. Figure below shows a descriptor-level deployment diagram for the box office system. This diagram shows the kinds of nodes in the system and the kinds of components they hold. A node is shown as a cube symbol.

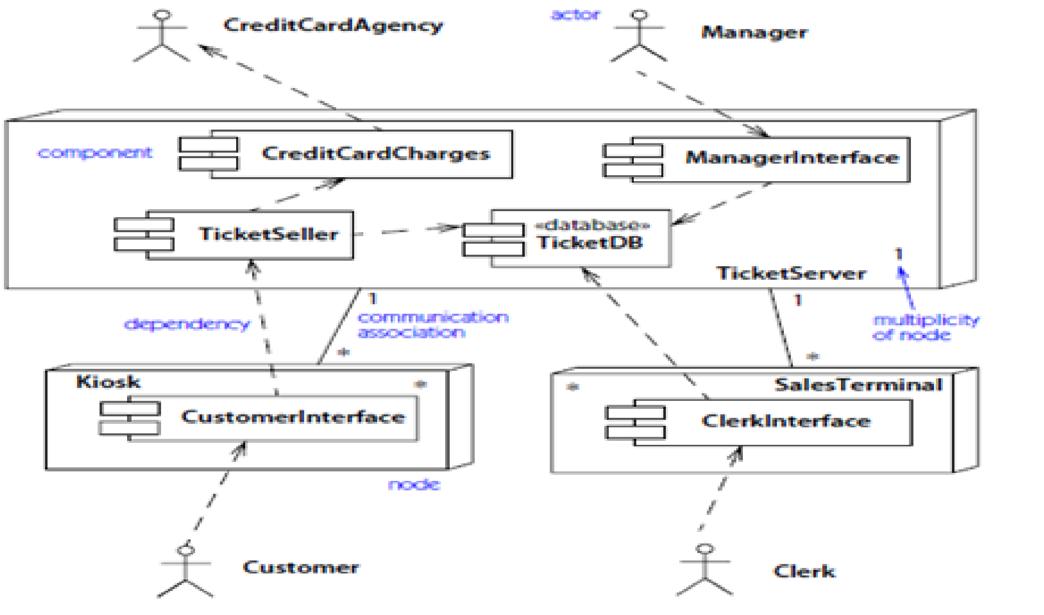

Figure *Deployment diagram (descriptor level)*

Figure below shows an instance-level deployment diagram for the box office system. The diagram shows the individual nodes and their links in a particular version of the system. The information in this model is consistent with the descriptor-level information in Figure above.

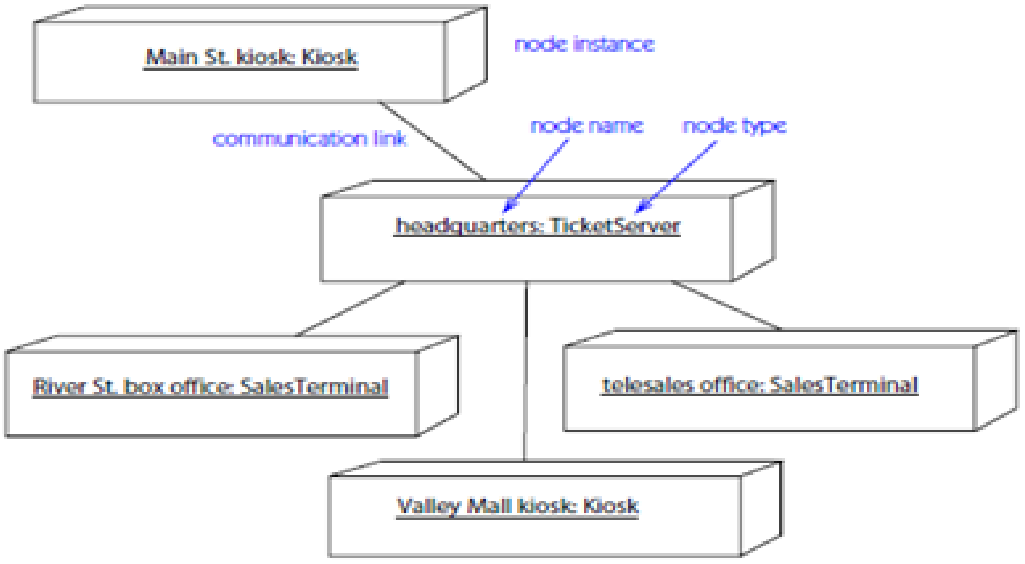

Figure *Deployment diagram (instance level)*

Model Management view

The model management view models the organization of the model itself. A model comprises a set of packages that hold model elements, such as classes, state machines, and use cases. Packages may contain other packages: therefore, a model designates a root package that indirectly contains all the contents of the model. Packages are units for manipulating the contents of a model, as well as units for access control and configuration control. Every model element is owned by one package or one other element.

A model is a complete description of a system at a given precision from oneviewpoint. There may be several models of a system from various viewpoints—forexample, an analysis model as well as a design model. A model is shown as a special kind of package.
A subsystem is another special package. It represents a portion of a system, witha crisp interface that can be implemented as a distinct component. Model management information is usually shown on class diagrams.Figure below shows the breakdown of the entire theater system into packages and their dependency relationships. The box office subsystem includes the previous examples in this chapter; the full system also includes theater operations and planning subsystems. Each subsystem consists of several packages.

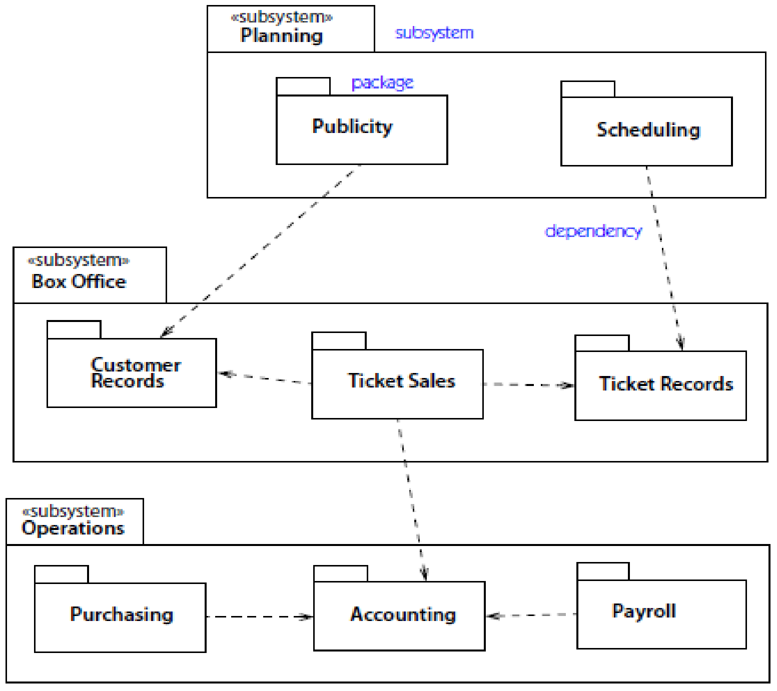

Figure *Packages*
