3 UML Walkthrough

This chapter presents a brief walkthrough of UML concepts and diagrams using a simple example. The purpose of the chapter is to organize the high-level UML concepts into a small set of views and diagrams that present the concepts visually. It shows how the various concepts are used to describe a system and how the views fit together. This summary is not intended to be comprehensive; many concepts are omitted. For more details, see the subsequent chapters that outline the UML semantic views, as well as the detailed reference material in the encyclopedia chapter.

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 various UML constructs in a brief space. It is deliberately simplified and is not presented in full detail. Presentation of a full model from an implemented system would neither fit in a small space nor highlight a sufficient range of constructs without excessive repetition.

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 division into different views is somewhat arbitrary, but we hope it is intuitive. One or two kinds 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.

Table 3-1: *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

Dynamic behavior describes the behavior of a system over time. Behavior can be 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 into hierarchical units. The package is the generic organizational unit for models. Spe-

cial 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, and tagged values. These constructs are applicable to elements of all views.

Table 3-1 shows the UML views and the diagrams that display them, as well as the main concepts relevant to each view. This table should not be taken as a rigid set of rules but merely as a guide to normal usage, as mixing of views is permitted.

Static View

The static view models concepts in the application domain, as well as internal concepts invented as part of the implementation of an application. This view is static because it does not describe the time-dependent behavior of the system, which is described in other views. The main constituents of the static view are classes and their relationships: association, generalization, and various kinds of dependency, such as realization and usage. A class is the description of a concept from the application domain or the application solution. Classes are the center around which the class view is organized; other elements are owned by or attached to classes. The static view is displayed in class diagrams, so called because their main focus is the description of classes.

Classes are drawn as rectangles. Lists of attributes and operations are shown in separate compartments. The compartments can be suppressed when full detail is not needed. A class may appear on several diagrams. Its attributes and operations are often suppressed on all but one diagram.

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.

Figure 3-1 shows a class diagram from the box office application. This diagram contains 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 are of 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.

Classes can be described at various levels of precision and concreteness. In the early stages of design, the model captures the more logical aspects of the problem. In the later stages, the model also captures design decisions and implementation details. Most of the views have a similar evolutionary quality.

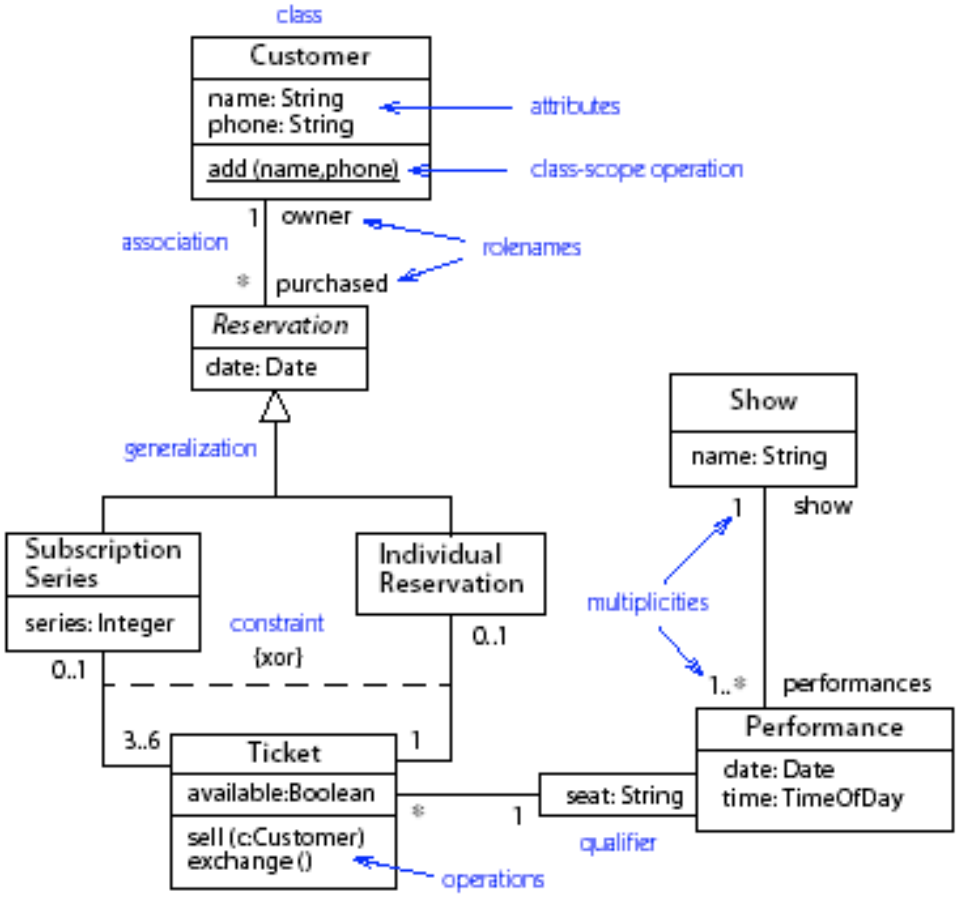

Figure 3-1. *Class diagram*

Use Case View

The use case view models the functionality of the system as perceived by outside users, called actors. A use case is a coherent unit of functionality expressed as a transaction among actors and the system. The purpose of the use case view is to list the actors and use cases and show which actors participate in each use case.

Figure 3-2 shows a use case diagram for the box office example. Actors include the clerk, supervisor, and kiosk. The kiosk is another system that accepts orders from 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 tick-

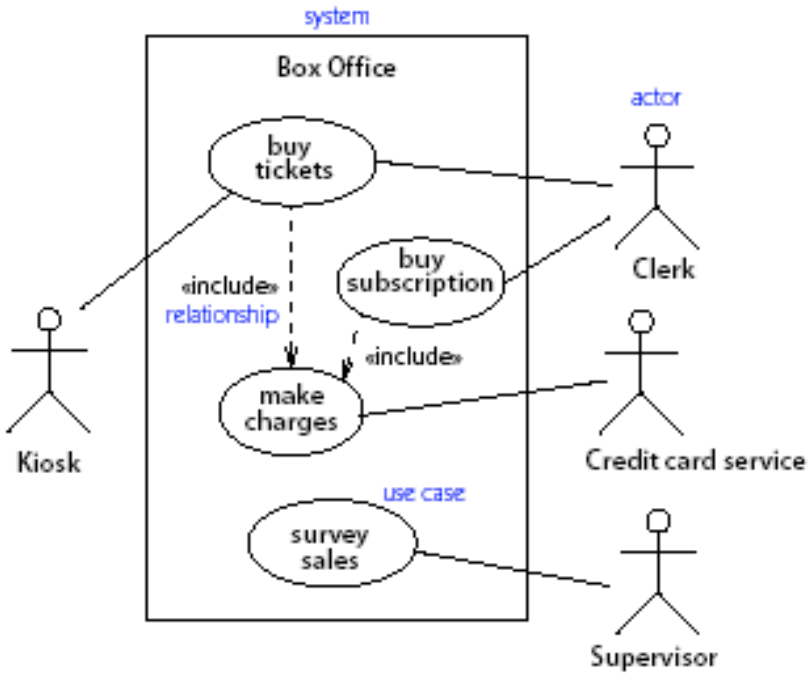

Figure 3-2. *Use case diagram*

ets and buying subscriptions include a common fragment—that is, making charges to the credit card service. (A complete description of a box office system would involve a number of other use cases, such as exchanging tickets and checking availability.)

Use cases can also be described at various levels of detail. They can be factored and described in terms of other, simpler use cases. A use case is implemented as a collaboration in the interaction view.

Interaction View

The interaction view describes sequences of message exchanges among roles that implement behavior of a system. A classifier role is the description of an object that plays a particular part within an interaction, as distinguished from other objects of the same class. This view provides a holistic view of behavior in a system—that is, it shows the flow of control across many objects. The interaction view is displayed in two diagrams focused on different aspects: sequence diagrams and collaboration diagrams.

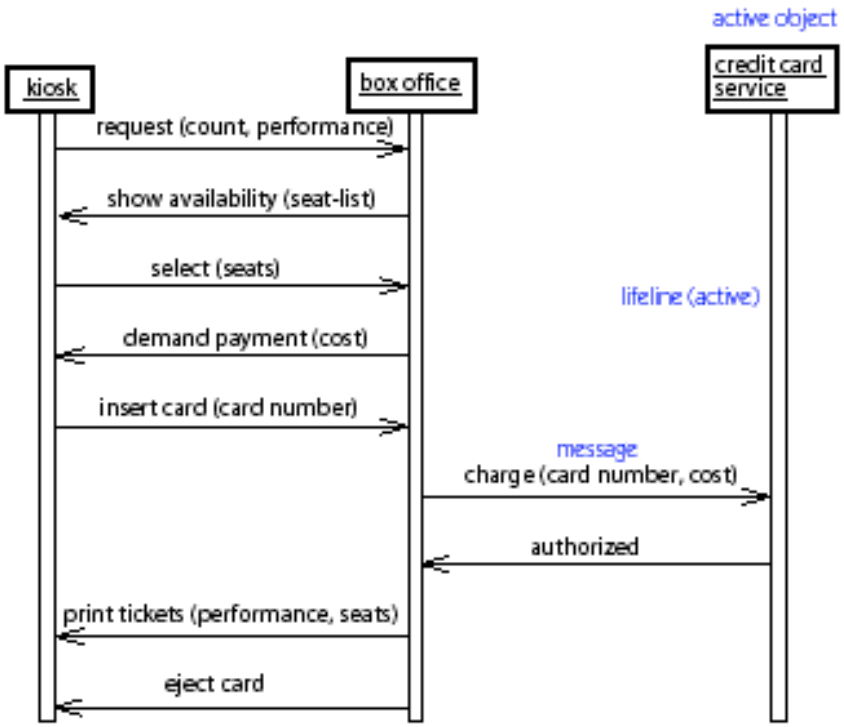

Figure 3-3. *Sequence diagram*

Sequence diagram

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 over time through the entire interaction. Messages are shown as arrows between lifelines. A sequence diagram can show a scenario—that is, an individual history of a transaction.

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 corresponds to an operation on a class or an event trigger on a transition in a state machine.

Figure 3-3 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. The steps for the make charges use case are included within the sequence, which involves communication with both the kiosk and the credit card service. This sequence diagram is at an early stage of development and does not show the full

details of the user interface. For example, the exact form of the seat list and the mechanism of specifying seats must still be determined, but the essential communication of the interaction has been specified by the use case.

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 the interaction. A classifier role describes an object and an association role describes a link within a collaboration. A collaboration diagram shows the roles in the interaction as a geometric arrangement (Figure 3-4). The messages are shown as arrows attached to the relationship lines connecting classifier roles. The sequence of messages 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,

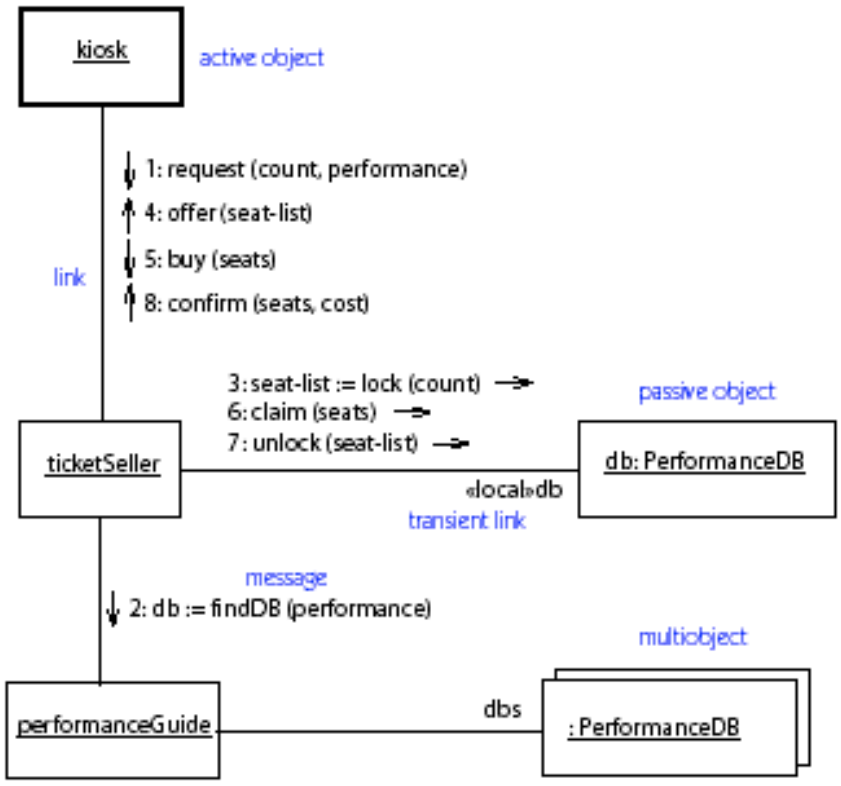

Figure 3-4. *Collaboration diagram*

as well as more permanent associations. When the behavior is implemented, the message sequencing corresponds to the nested calling structure and signal passing of the program.

Figure 3-4 shows a collaboration diagram for the reserve tickets interaction at a later stage of development. The collaboration shows the interaction among internal objects in the application to reserve tickets. The request arrives from the kiosk and is used to find the database for the particular performance from the set of all performances. The pointer db that is returned to the ticketSeller object represents a local transient link to a performance database that is maintained during the interaction and then discarded. The ticket seller requests a number of seats to the performance; a selection of seats in various price ranges is found, temporarily locked, and returned to the kiosk for the customer's selection. When the customer makes a selection from the list of seats, the selected seats are claimed and the rest are unlocked.

Both sequence diagrams and collaboration diagrams show interactions, but they emphasize different aspects. A sequence diagram shows time sequence as a geometric dimension, but the relationships among roles are implicit. A collaboration diagram shows the relationships among roles geometrically and relates messages to the relationships, but time sequences are less clear because they are implied by the sequence numbers. Each diagram should be used when its main aspect is the focus of attention.

State Machine View

A state machine models the possible life histories of an object of a class. A state machine contains states connected by transitions. Each state models a period of time during the life of an object during which it satisfies certain conditions. When an event occurs, it may cause the firing of a transition that takes the object to a new state. When a transition fires, an action attached to the transition may be executed. State machines are shown as statechart diagrams.

Figure 3-5 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 tickets purchased 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 customer takes 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, and other reactive subsystems. They may also be used to describe passive objects that go through several qualitatively distinct phases during their lifetime, each of which has its own special behavior.

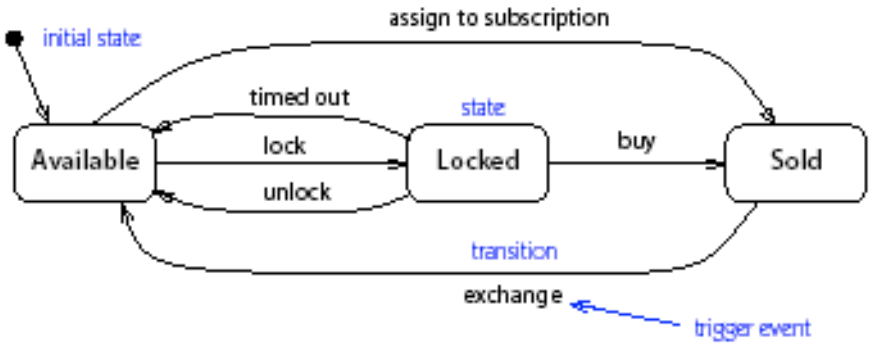

Figure 3-5. *Statechart diagram*

Activity View

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 describes both sequential and concurrent groups of activities. Activity graphs are shown on activity diagrams.

Figure 3-6 shows an activity diagram for the box office. This diagram shows the activities involved in mounting a show. (Don't take this example too seriously if you have theater experience!) Arrows show sequential dependencies—for example, shows must be picked before they are scheduled. Heavy bars show forks or joins 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 relationships connecting the action and an object flow state.

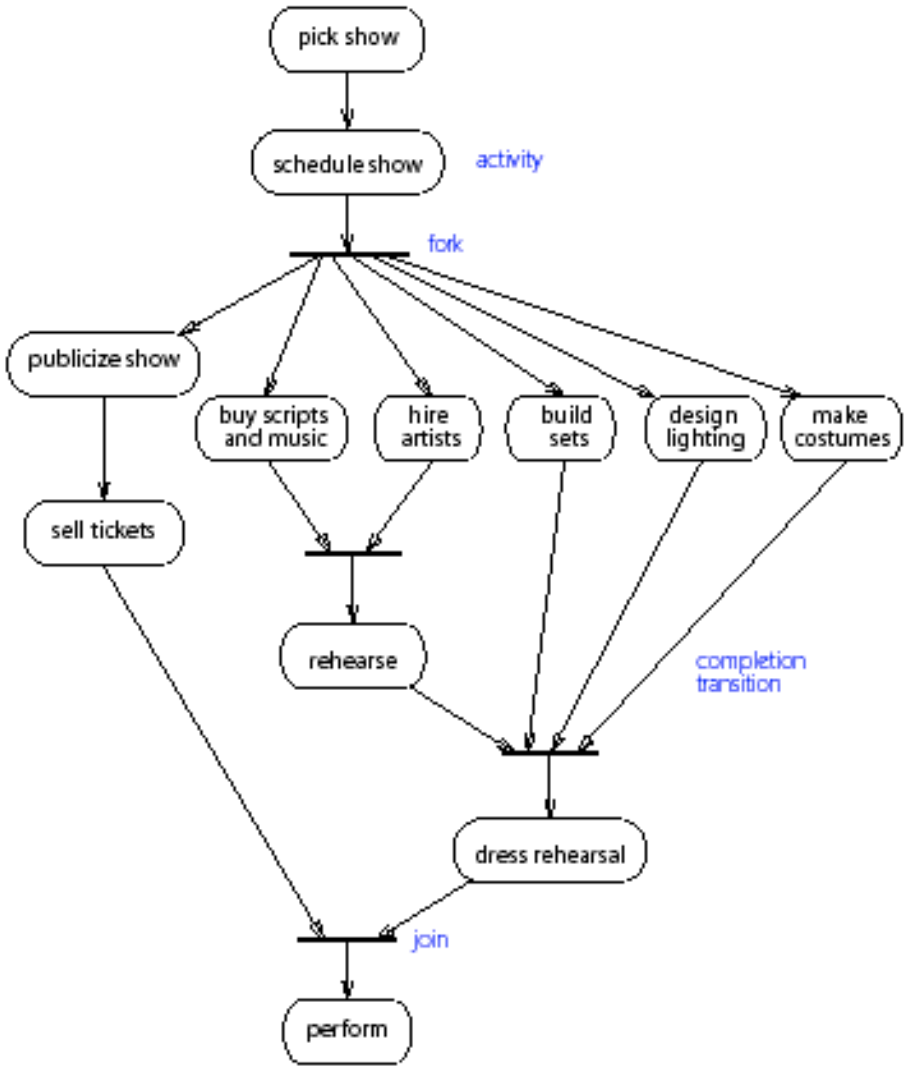

Figure 3-6. *Activity diagram*

Physical Views

The previous views model the concepts in the application from a logical viewpoint. 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 3-7 shows a component diagram for the box office system. There are three user

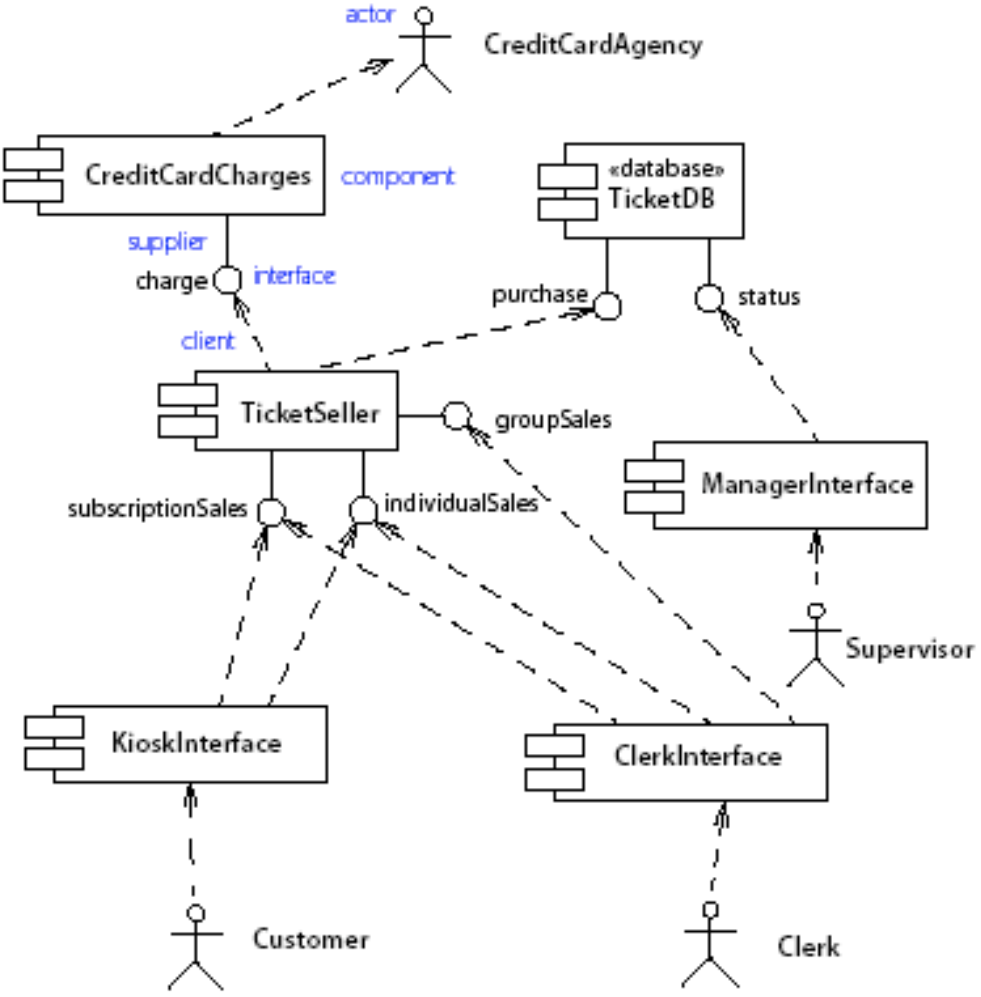

Figure 3-7. *Component diagram*

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 ser-

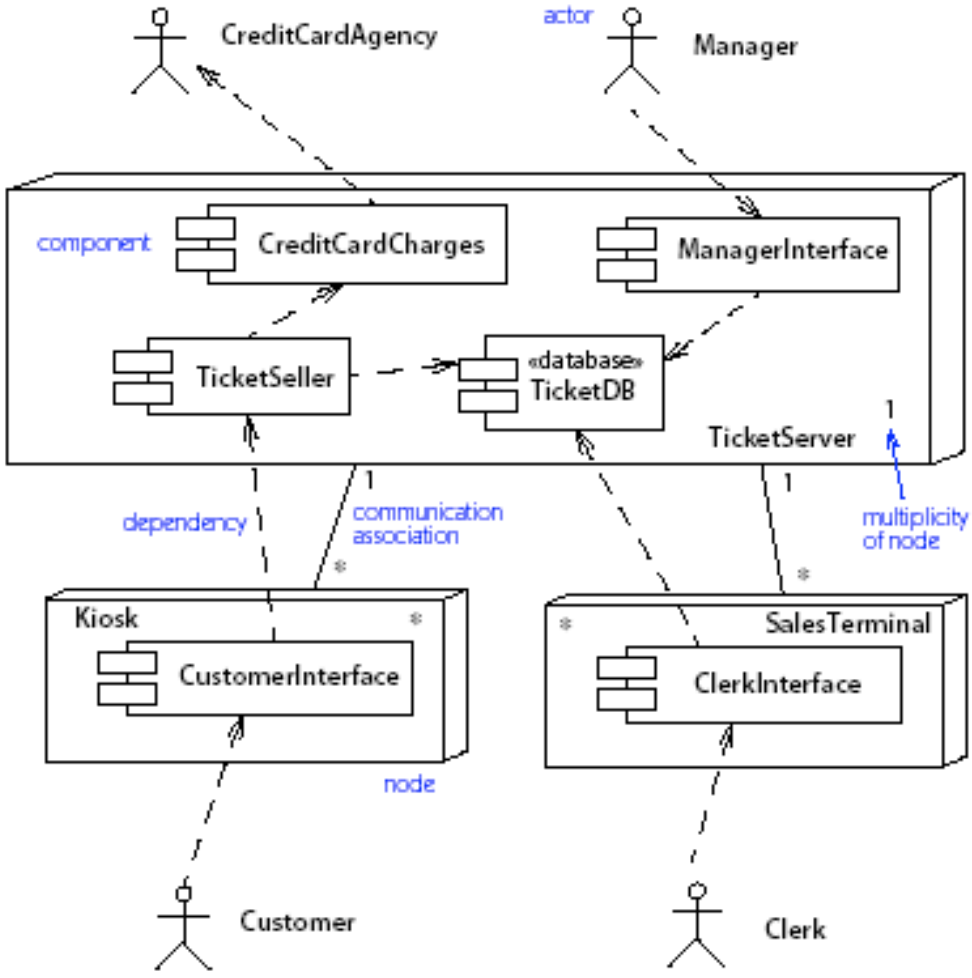

Figure 3-8. *Deployment diagram (descriptor level)*

vices 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.

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 3-8 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 3-9 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 3-8.

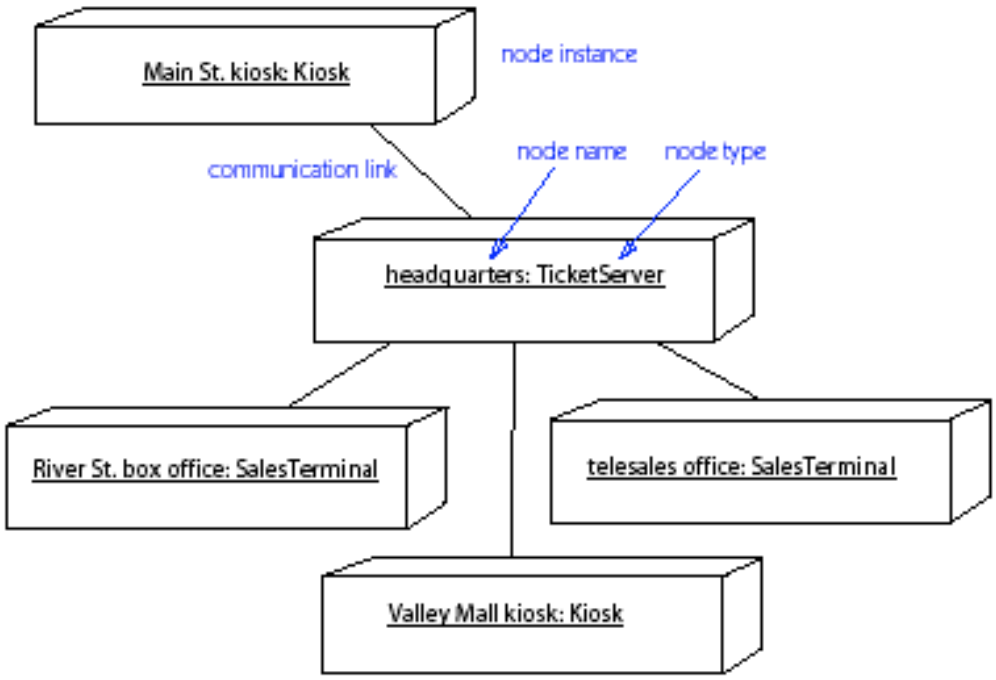

Figure 3-9. *Deployment diagram (instance level)*

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.

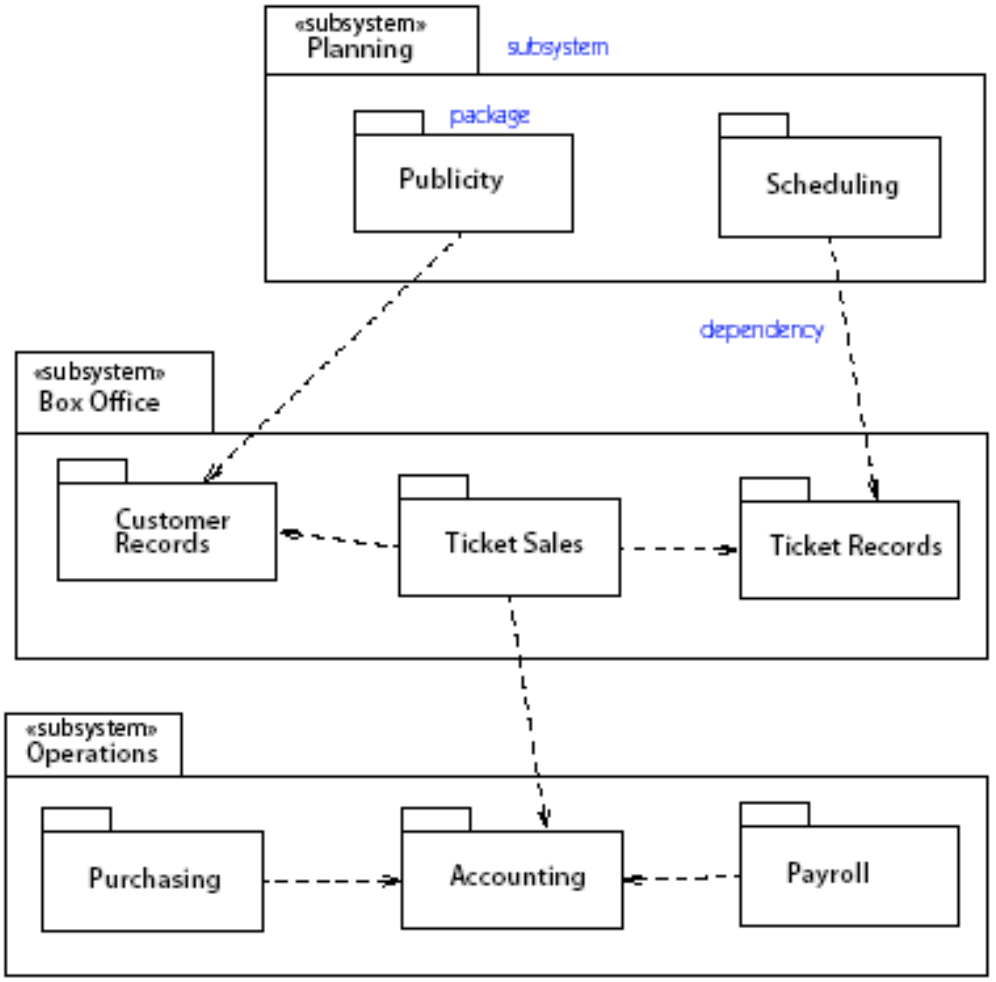

Figure 3-10. *Packages*

A model is a complete description of a system at a given precision from one viewpoint. There may be several models of a system from various viewpoints—for example, 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, with a crisp interface that can be implemented as a distinct component.

Model management information is usually shown on class diagrams.

Figure 3-10 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.

Extensibility Constructs

UML includes three main extensibility constructs: constraints, stereotypes, and tagged values. A constraint is a textual statement of a semantic relationship expressed in some formal language or in natural language. A stereotype is a new kind of model element devised by the modeler and based on an existing kind of model element. A tagged value is a named piece of information attached to any model element.

These constructs permit many kinds of extensions to UML without requiring changes to the basic UML metamodel itself. They may be used to create tailored versions of the UML for an application area.

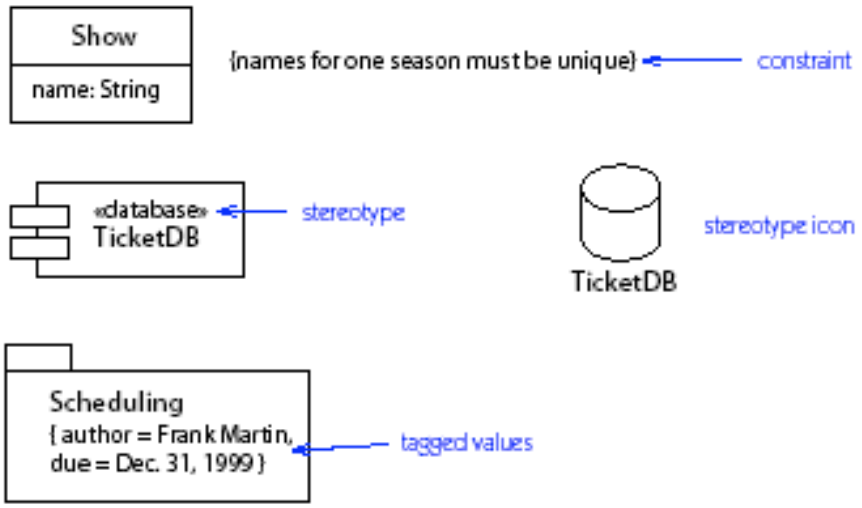

Figure 3-11. *Extensibility constructs*

Figure 3-11 shows examples of constraints, stereotypes, and tagged values. The constraint on class Show ensures that the names of shows are unique. Figure 3-1 shows an xor constraint on two associations; an object can have a link from one of them at a time. Constraints are useful for making statements that can be expressed in a text language but which are not directly supported by UML constructs.

The stereotype on component TicketDB indicates that the component is a database, which permits the interfaces supported by the component to be omitted as they are the interfaces supported by all databases. Modelers can add new stereotypes to represent special elements. A set of implied constraints, tagged values, or code generation properties can be attached to a stereotype. A modeler can define an icon for a given stereotype name as a visual aid, as shown in the diagram. The textual form may always be used, however.

The tagged values on package Scheduling show that Frank Martin is responsible for finishing it before the end of the millennium. Any arbitrary piece of information can be attached to a model element as a tagged value under a name chosen by the modeler. Text values are especially useful for project management information and for code generation parameters. Most tagged values would be stored as pop-up information within an editing tool and would not usually be displayed on printed pictures.

Connections Among Views

The various views coexist within a single model and their elements have many connections, some of which are shown in Table 3-2. This table is not meant to be complete, but it shows some of the major relationships among elements from different views.

Table 3-2: *Some Relationships Among Elements in Different Views*

Element	*Element*	*Relationship*
class	state machine	ownership
operation	interaction	realization
use case	collaboration	realization
use case	interaction instance	sample scenario
component instance	node instance	location
action	operation	call
action	signal	send
activity	operation	call
message	action	invocation
package	class	ownership
role	class	classification