Object-Oriented Software Modeling Using UML

Lecture Notes

UML Software Models

Taewoong Jeon

Jeon@korea.ac.kr

Dept. Computer Science, Korea University

May 3, 2014

Overview

[RJB05]

An objected-oriented 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.

The UML can be used to capture information about the static structure and dynamic behavior of a system.

UML Concept Areas (1/6)

[RJB05]

UML concepts and models can be grouped into the following concept areas:

- Static structure view
- Design view
- Deployment view
- Dynamic behavior view
- Model organization

UML Concept Areas (2/6)
Static View

[RJB05]

Application concepts are modeled as classes, each of which describes discrete objects that hold information and communicate to implement behavior.

- The information they hold is modeled as attributes.
- The behavior they perform is modeled as operations.

Object-to-object relationships are modeled as associations among classes.

Several classes can share their common structure and behavior using generalization.

The static view is notated using class diagrams and its variants.

UML Concept Areas (3/6)
Design View

[RJB05]

UML models are meant for both logical analysis and designs intended for implementation. Certain constructs represent design items.

A structured classifier expands a class into its implementation as a collection of parts held together by connectors.

A class can encapsulate its internal structure behind externally visible ports.

A collaboration models a collection of objects that play roles within a transient context.

A component is a replaceable part of a system that conforms to and provides the realization of a set of interfaces.

UML Concept Areas (4/6)
Deployment View

[RJB05]

A node is a run-time computing resource that defines a location.

An artifact is a physical unit of information or behavior description in a computing system.

- Artifacts are deployed on nodes.
- An artifact can be a manifestation (i.e., an implementation) of a component.

The deployment view describes the configuration of nodes in a running system, and the arrangement of artifacts on them.

UML Concept Areas (5/6)
Behavior View

[RJB05]

There are three ways to model behavior:

- The life history of one object as it interacts with the rest of the world (state machine view)
- The communication patterns of a set of connected objects as they interact to implement behavior (interaction view)
- The execution process of a computation as it passes through various activities (activity view)

UML Concept Areas (6/6)
Model Organization

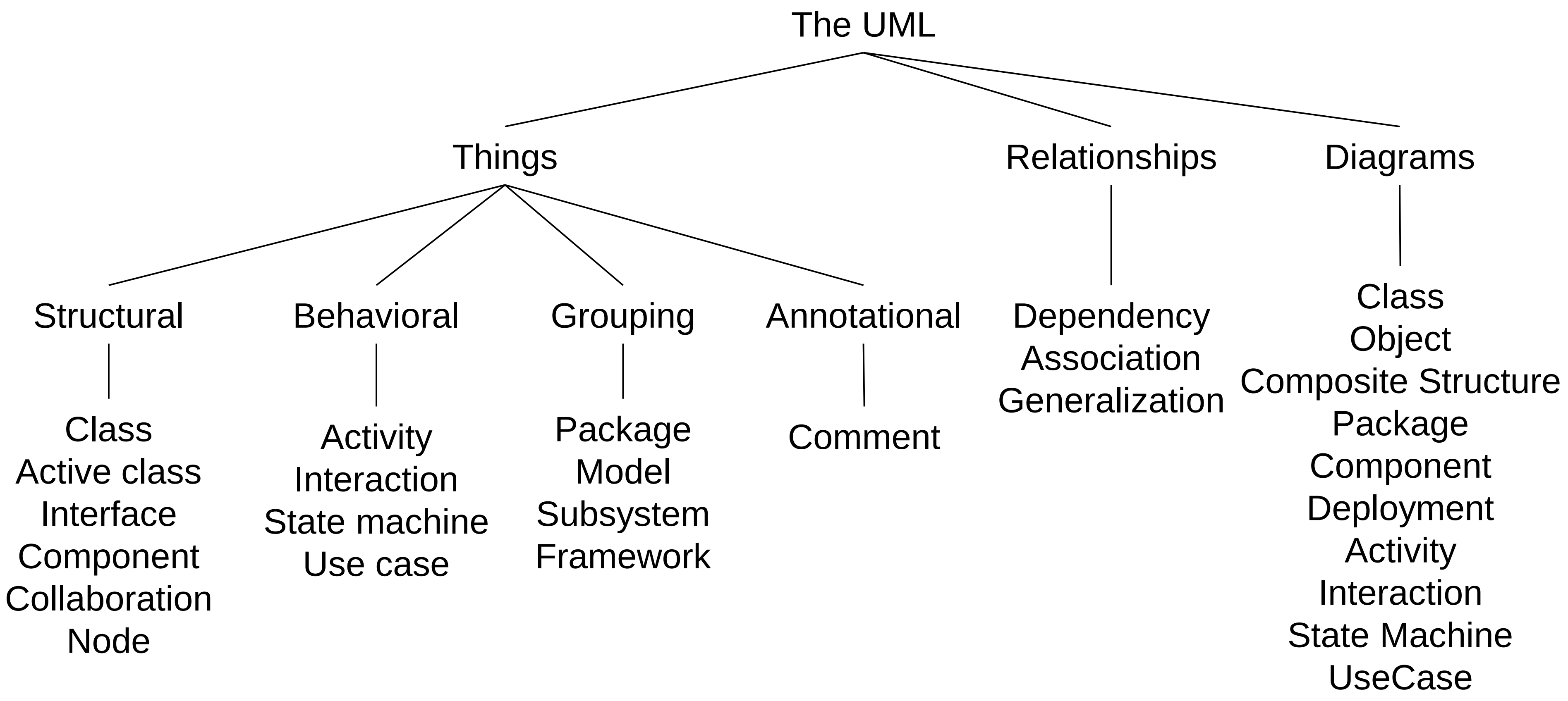
[RJB05]

In a large system, the modeling information must be divided into coherent pieces so that teams can work on different parts concurrently.

Packages are general-purpose hierarchical organizational units of UML models.

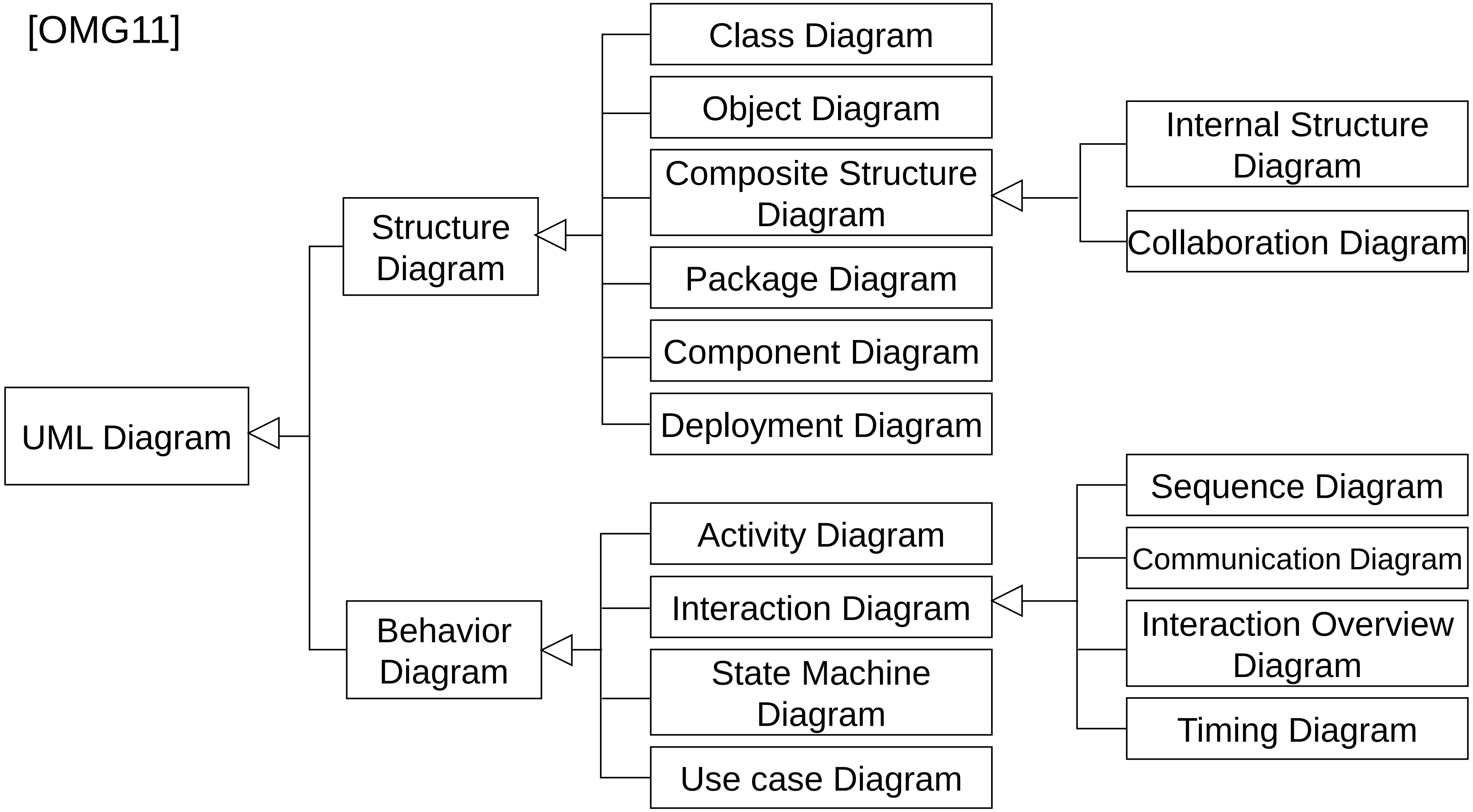
> They can be used for storage, access control, configuration management, and constructing libraries that contain reusable model fragments.

Vocabulary of UML 2



Taxonomy of UML 2 Diagrams

[OMG11]



Use Case View (1/2)

[RJB05]

The use case view models the functionality of a subject (e.g., a system) as perceived by outside agents, called actors, that interact with the subject from a particular viewpoint.

A use case is a unit of functionality expressed as a transaction among actors and the subject.

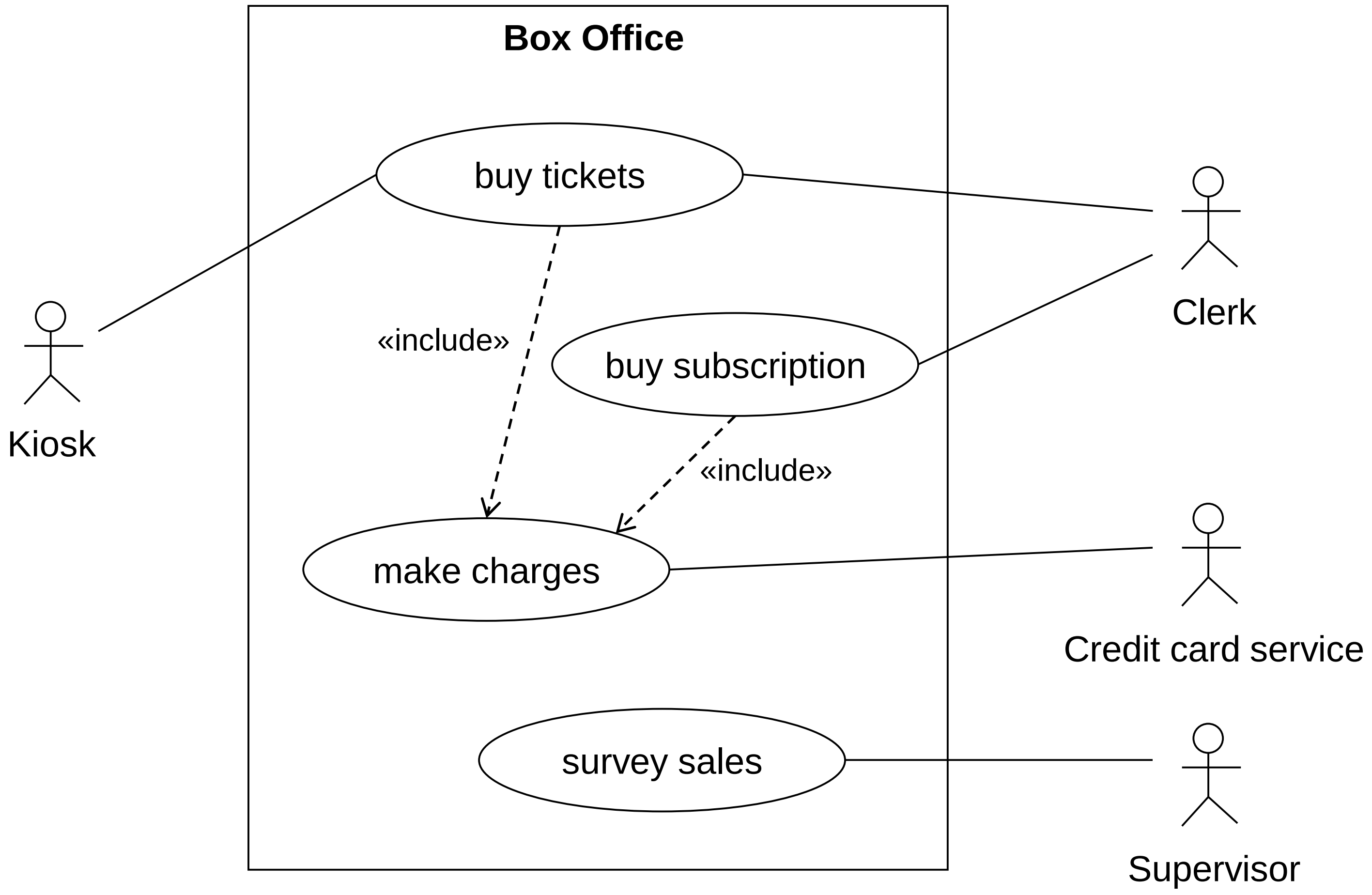
The purpose of the use case view is to list the actors and use cases and show which actors participate in each use case.

The behavior of use cases is expressed using dynamic views, particularly the interaction view.

Use Case View (2/2)

[RJB05]

A Use Case Diagram



Static View (1/4)

[RJB05]

The elements of the static view of a model are concepts that are meaningful in an application, including all kinds of found in systems.

- Real-world concepts.
- Abstract concepts.
- Implementation concepts.
- Computer concepts.

For example, a ticket system for a theater has concepts such as tickets, reservations, subscription plans, seat assignment algorithms, interactive web pages for ordering, and archival data for redundancy.

Static View (2/4)

[RJB05]

The static view captures object structure.

An object-oriented system unifies data structure and behavioral features into a single object structure.

In the object-oriented perspective, data and behavior are closely related.

For example, a Ticket object carries data, such as its price, date of performance, and seat number, as well as operations on it, such as reserving itself or computing its price with a special discount.

Static View (3/4)

[RJB05]

The static view describes behavioral declarations, such as operations, as discrete things to be named, owned by classes, and invoked.

Their dynamic behavior is described by other views that describe the internal details of their dynamics.

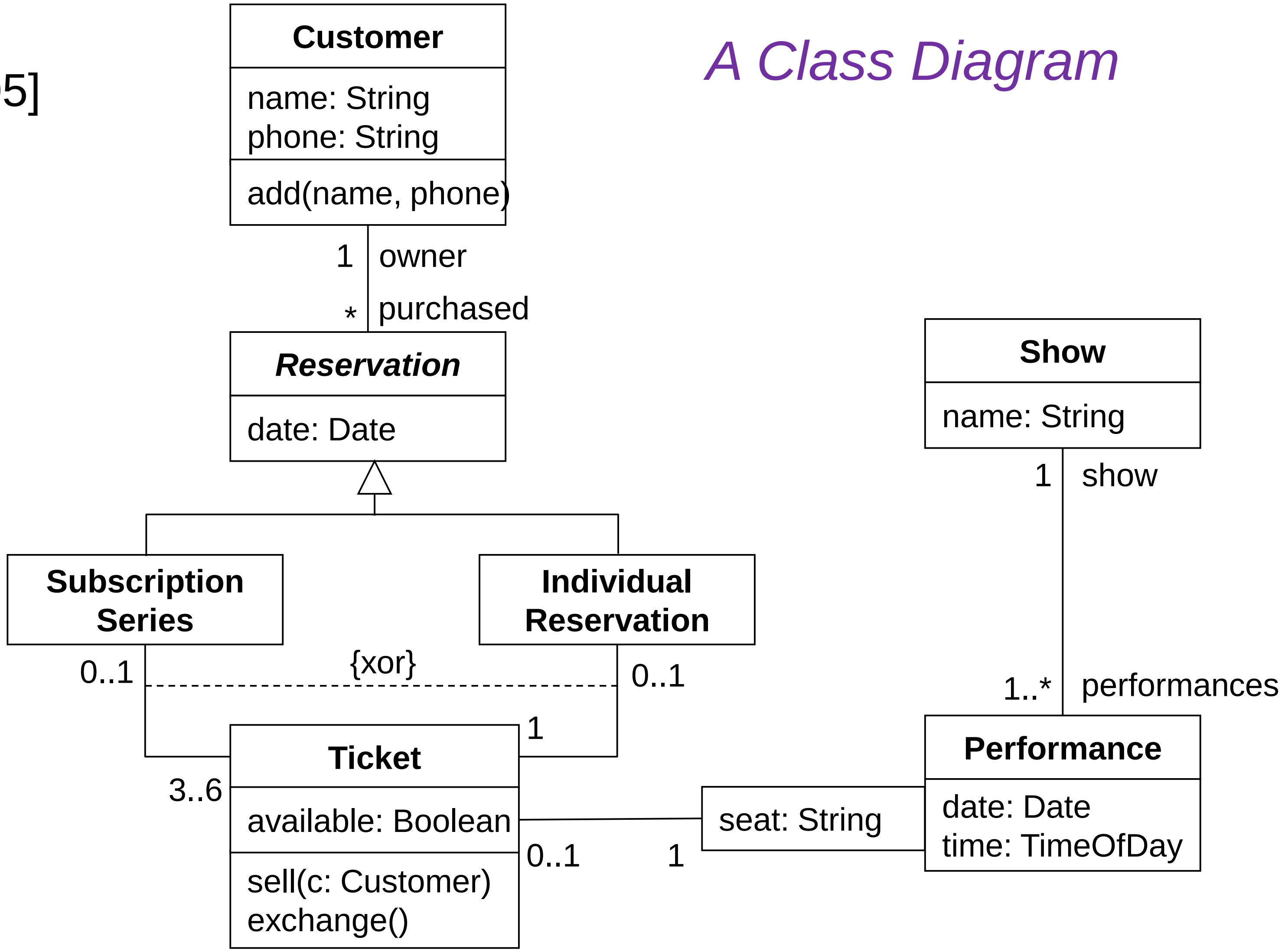
These other views include the interaction view and the state machine view.

Dynamic views require the static view to describe the things that interact dynamically. The static view is the foundation on which the other views are built.

Static View (4/4)

[RJB05]

A Class Diagram



Design View (1/9)

[RJB05]

The design view shows decisions about decomposition of a system into modular units with encapsulation boundaries and external interfaces.

Although the elements in the design view are more abstract than the final code, they do require knowledge of implementation trade-offs that will eventually be reflected in the code.

Design View (2/9)

[RJB05]

Complex systems require multiple levels of structure.

- During early modeling, a class is defined by its external properties.
- During design modeling, the internal design of a high-level class may be expanded into constituent parts.

A structured classifier is a classifier with internal parts that are connected within the context of the classifier.

The types of the internal parts may themselves be structured classifiers. Therefore the decomposition of the system can span several levels.

Design View (3/9)

Collaborations (1/3)

[RJB05]

In a design, independent objects often work together to perform operations and other behaviors.

A collaboration is a description of a group of objects than have a temporary relationships within the context of performing a behavior.

The connections among objects in a collaboration may include various kinds of transient relationships, such as parameters, variables, and derived relationships, as well as ordinary associations.

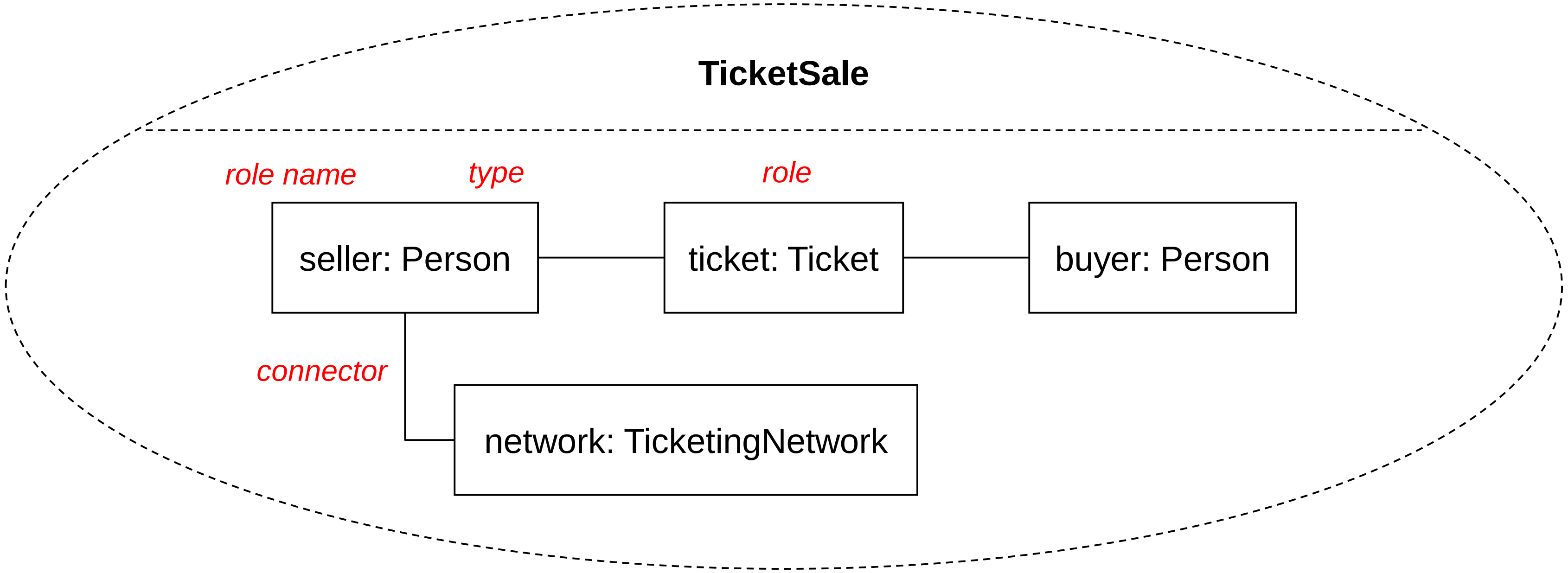
Design View (4/9)

Collaborations (2/3)

[RJB05]

A Ticket in a TicketSale collaboration has a seller, something that is not relevant to a Ticket in general.

One person may be a buyer in one collaboration and a seller in another collaboration.

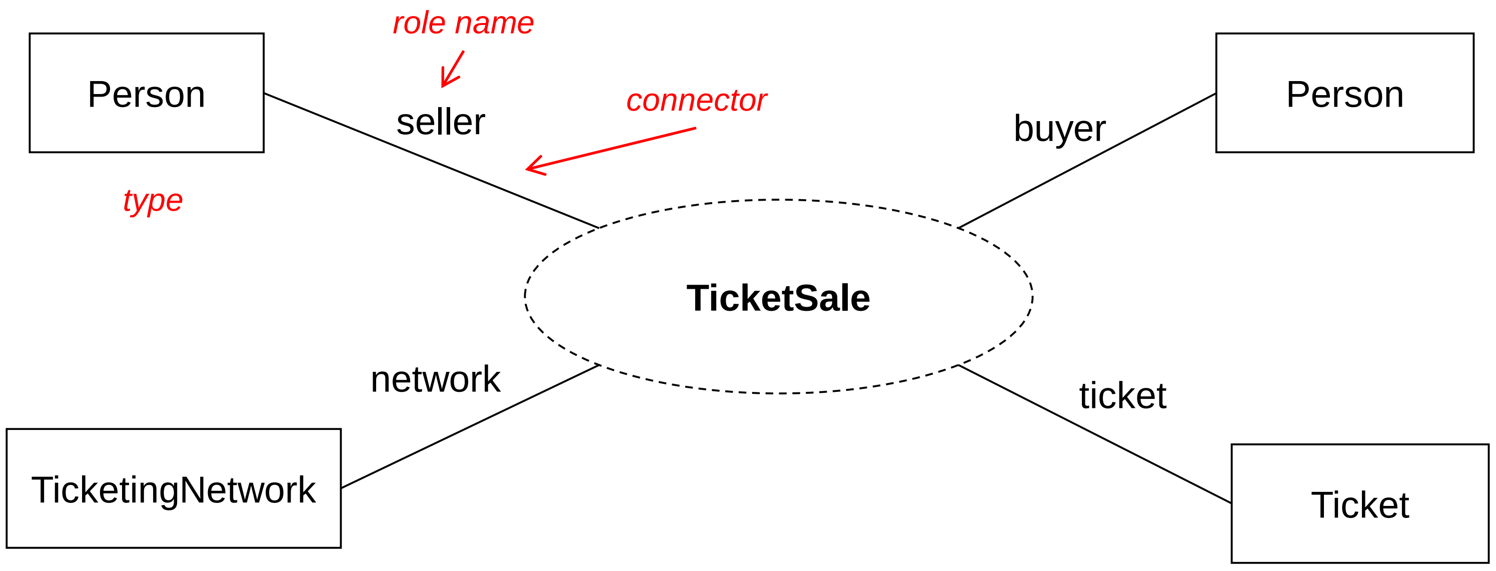


Design View (5/9)

Collaborations (3/3)

[OMG11]

Using an alternative notation, a line may be drawn from the collaboration icon to each symbols denoting classifiers that are the types of properties of the collaboration.



Design View (6/9)
Components (1/4)

[RJB05]

The design view shows the logical organization of the reusable pieces of the system into substitutable units, called components.

A component has a set of external interfaces and a hidden, internal implementation.

Components interact through interfaces so that dependencies on specific other components are avoided.

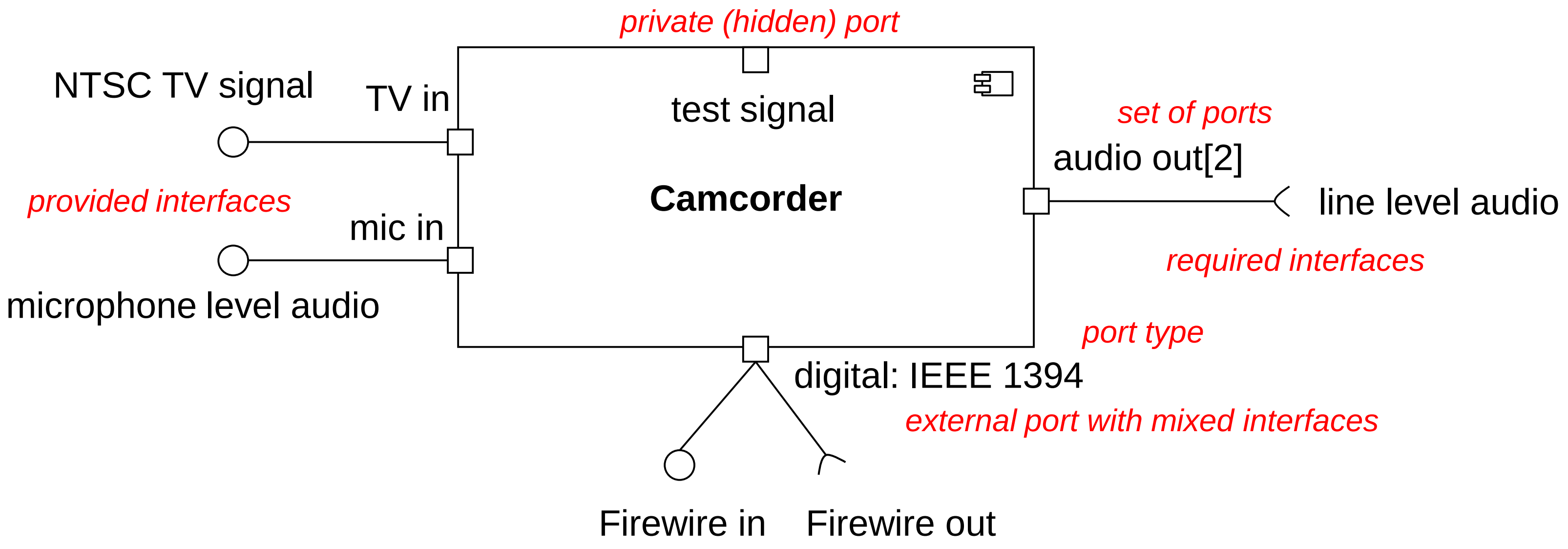
During implementation, any component that supports an interface can be substituted for it, allowing parts of a system to be developed without dependency on internal implementation.

Design View (7/9)

Components (2/4)

[RJB05]

External view of a component

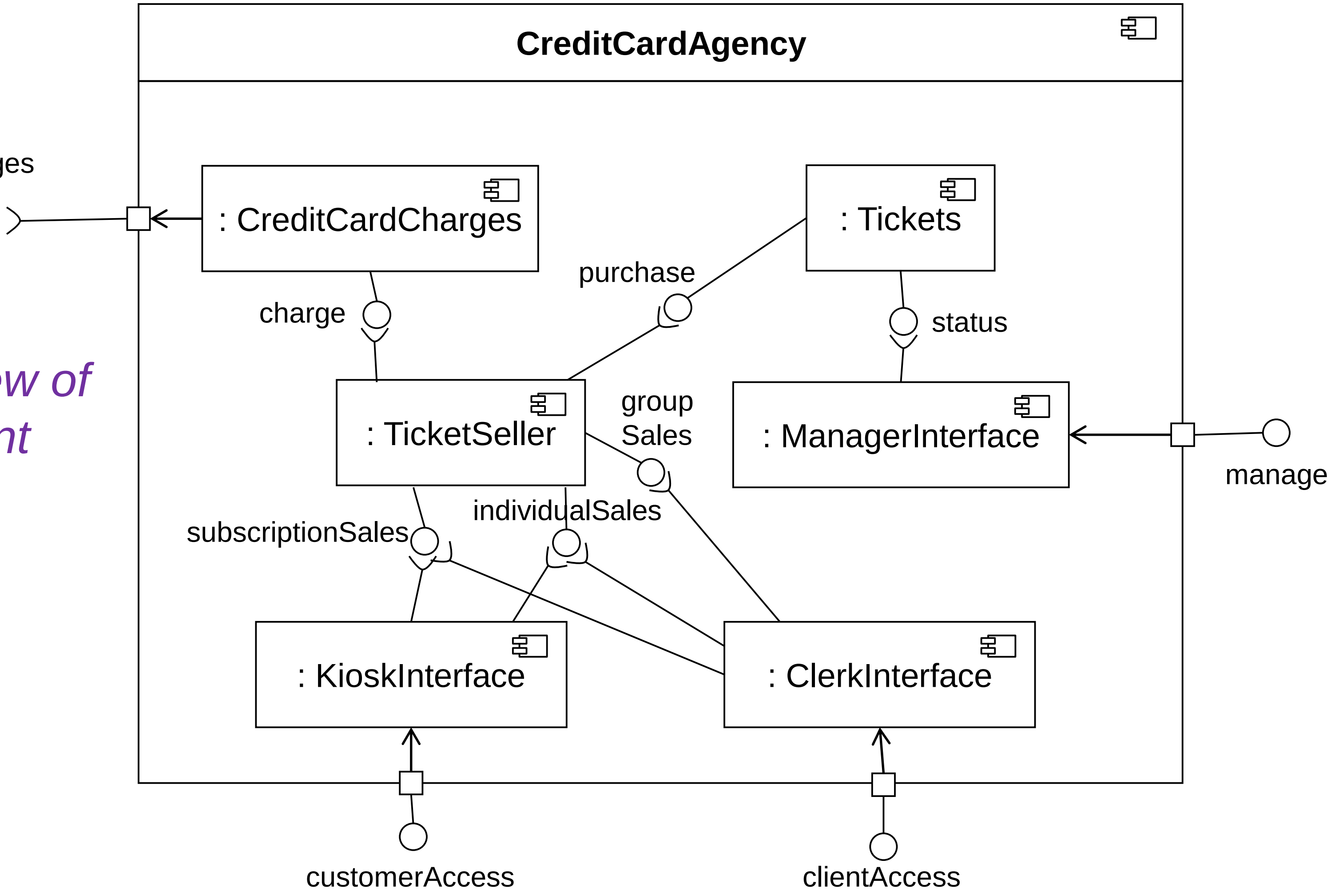


Design View (8/9)

Components (3/4)

[RJB05]

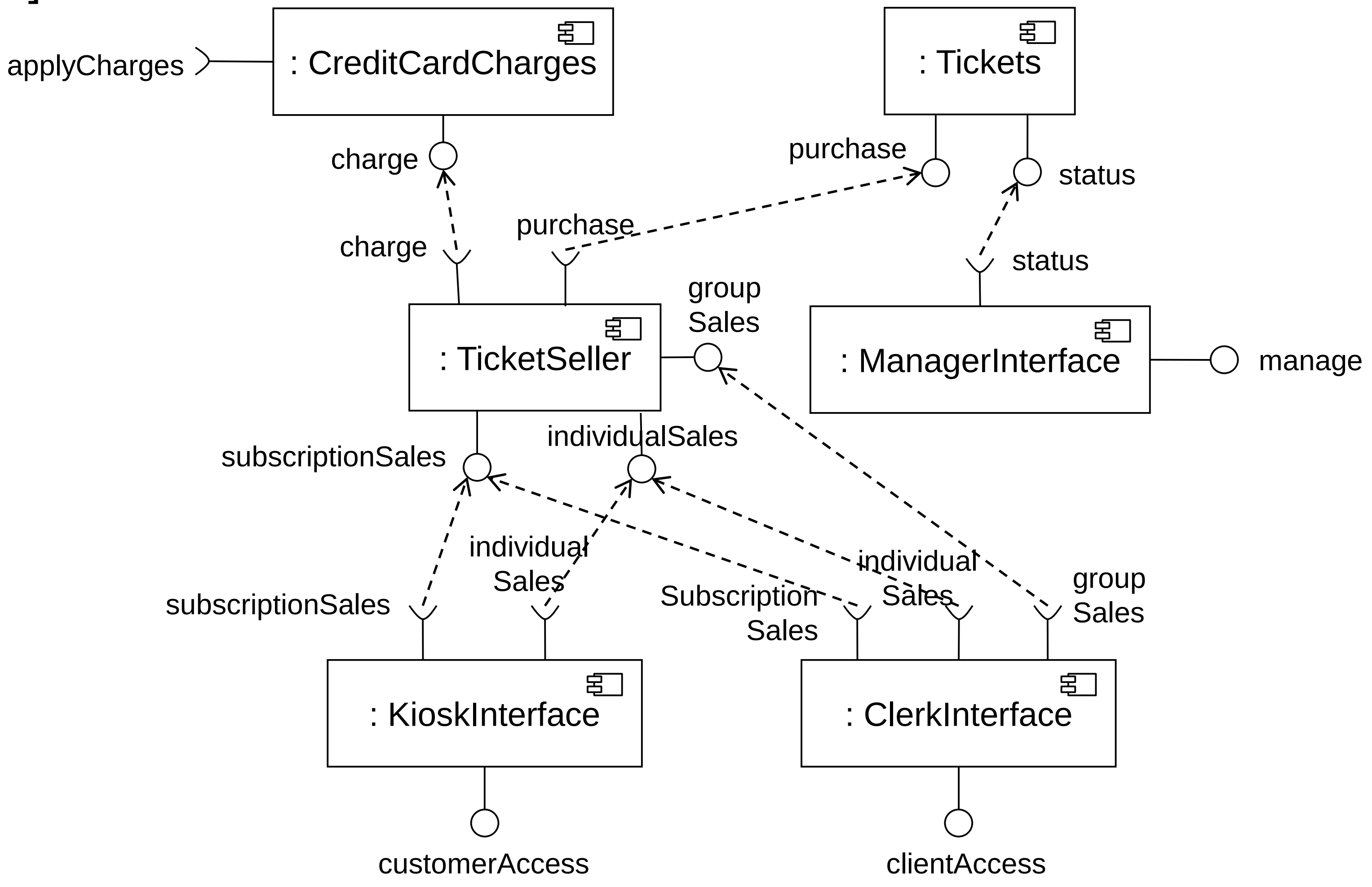
External view of a component



Design View (9/9)

Component Diagram (4/4)

[RJB05]



Behavior View (1/2)

[OMG11]

A variety of behavior specification mechanisms are supported by UML.

- Automata (state machines)
- Petri-net like graphs (activity diagrams)
- Partially-ordered sequences of event occurrences (interactions)

Behavior View (2/2)

[OMG11]

The styles of behavioral specification differ in their expressive power and domain of applicability.

> Further, they may specify behaviors explicitly, by describing the observable event occurrences resulting from the execution of the behavior,
>
> or implicitly, by describing a machine that would induce these events.

The choice of specification mechanism is one of convenience and purpose; typically the same kind of behavior could be described by any of the different mechanisms.

State Machine View (1/2)

[RJB05]

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 which satisfies certain conditions.
- When an event occurs, it may cause the firing of a transition from the current state to a new state.
- When a transition fires, an effect (action or activity) attached to the transition may be executed.

State machines are shown as state machine diagrams.

State Machine View (2/2)

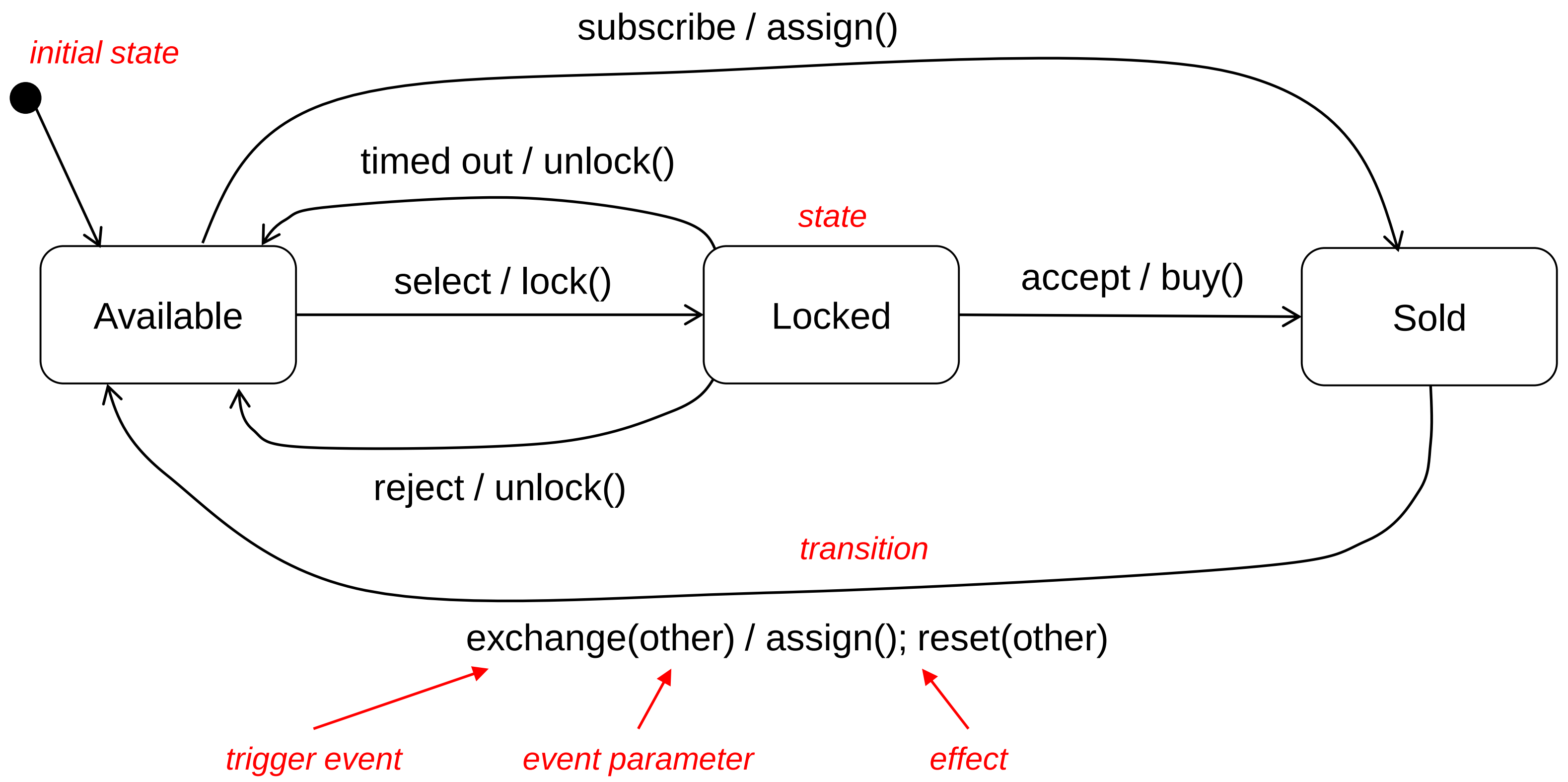
[RJB05]

State machines may be used to describe:

- reactive subsystems such as user interfaces and device controllers.
- passive objects that go through several qualitatively distinct phases during their lifetime, each of which has its own special behavior.

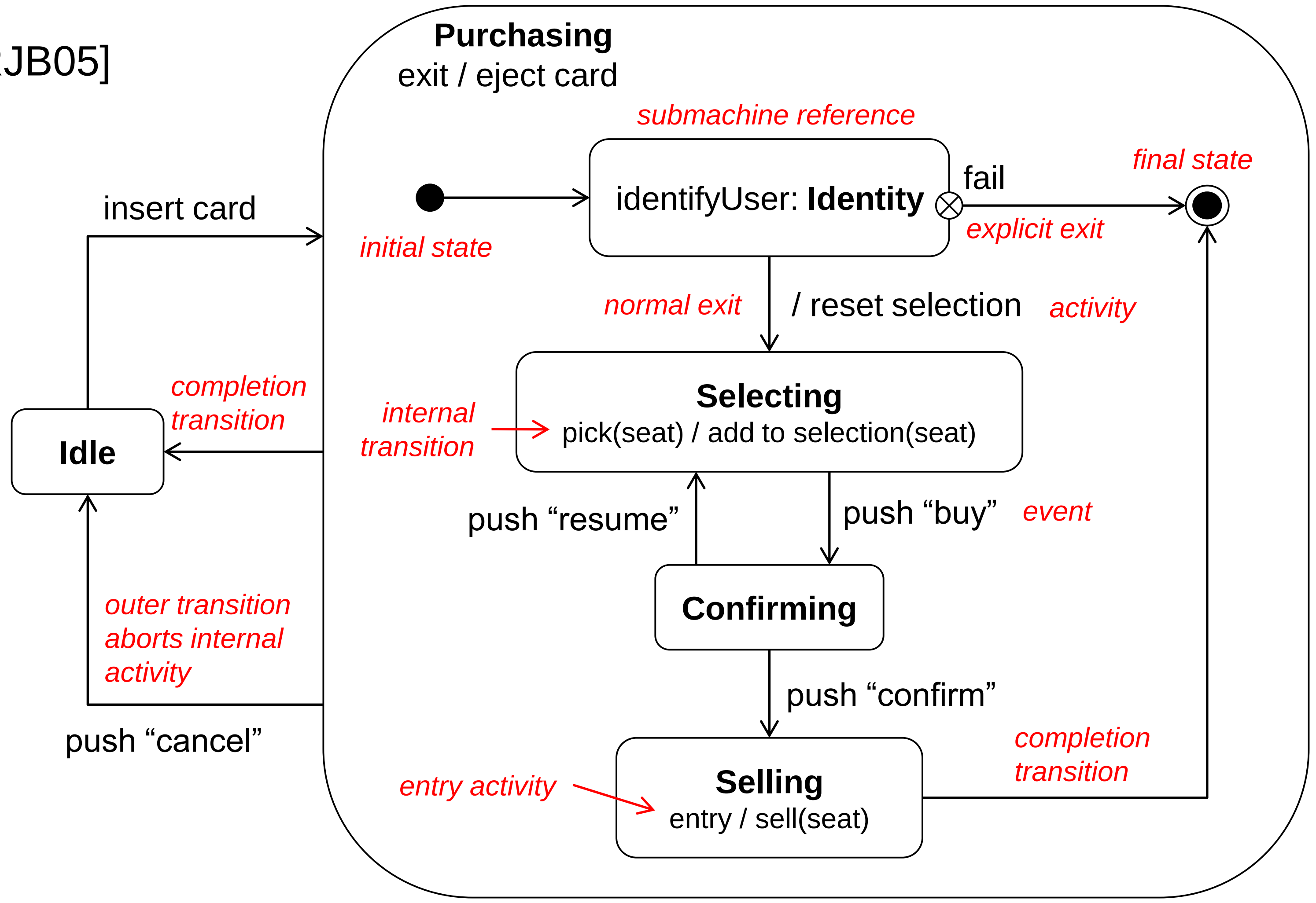
State Machine Diagram

[RJB05]



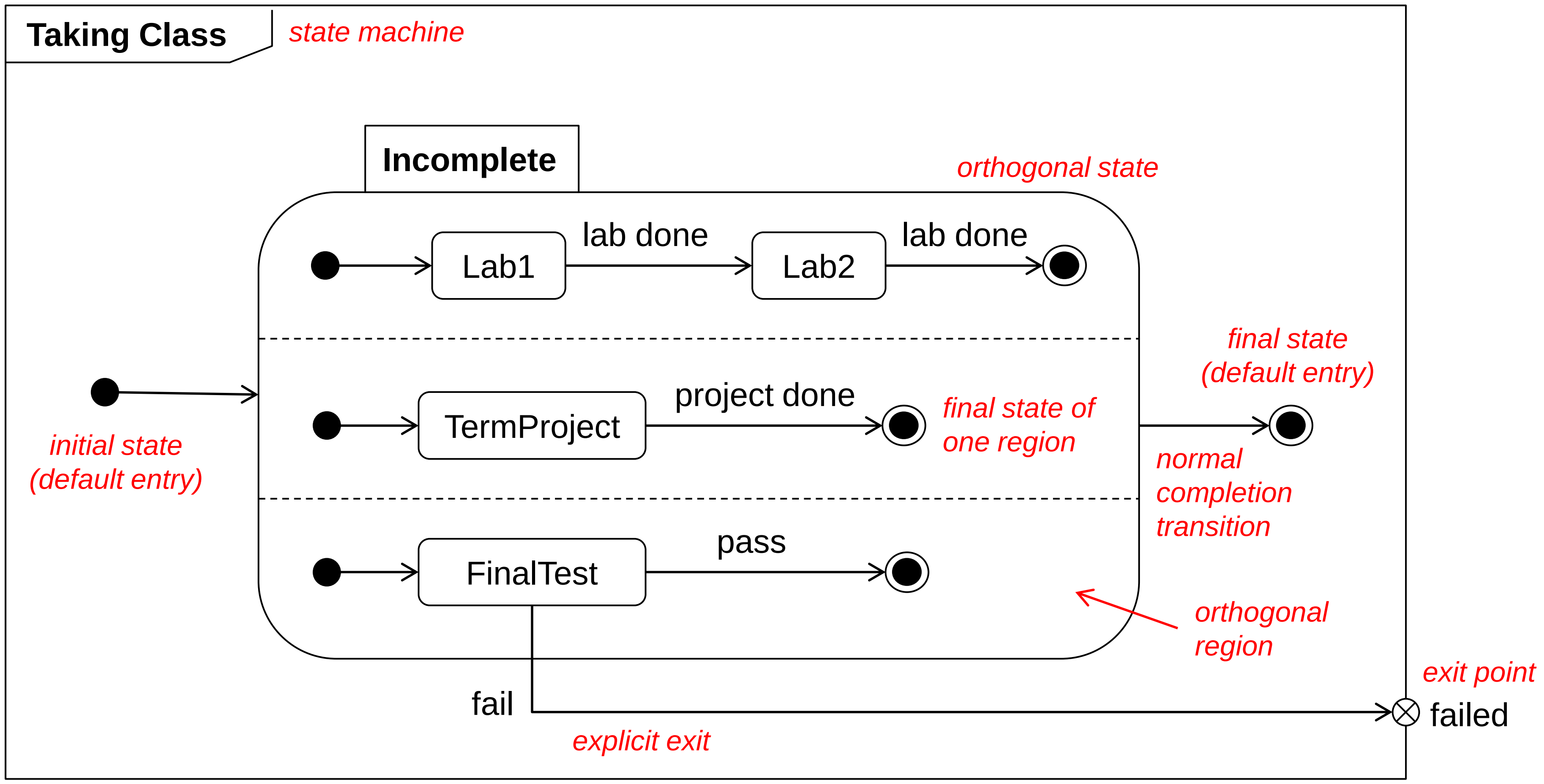
State Machine Containing a Sequential Decomposition of a State

[RJB05]



State Machine with Orthogonal Composite State

[RJB05]



Activity View (1/2)

[RJB05]

An activity shows the flow of control among the computational activities involved in performing a task or a workflow.

Activities are shown on activity diagrams.

- An action is a primitive computational step.
- An activity node is a group of actions or subactivities.
- An activity describes both sequential and concurrent computation.

Activity View (2/2)

[RJB05]

Activity diagrams may be used to model:

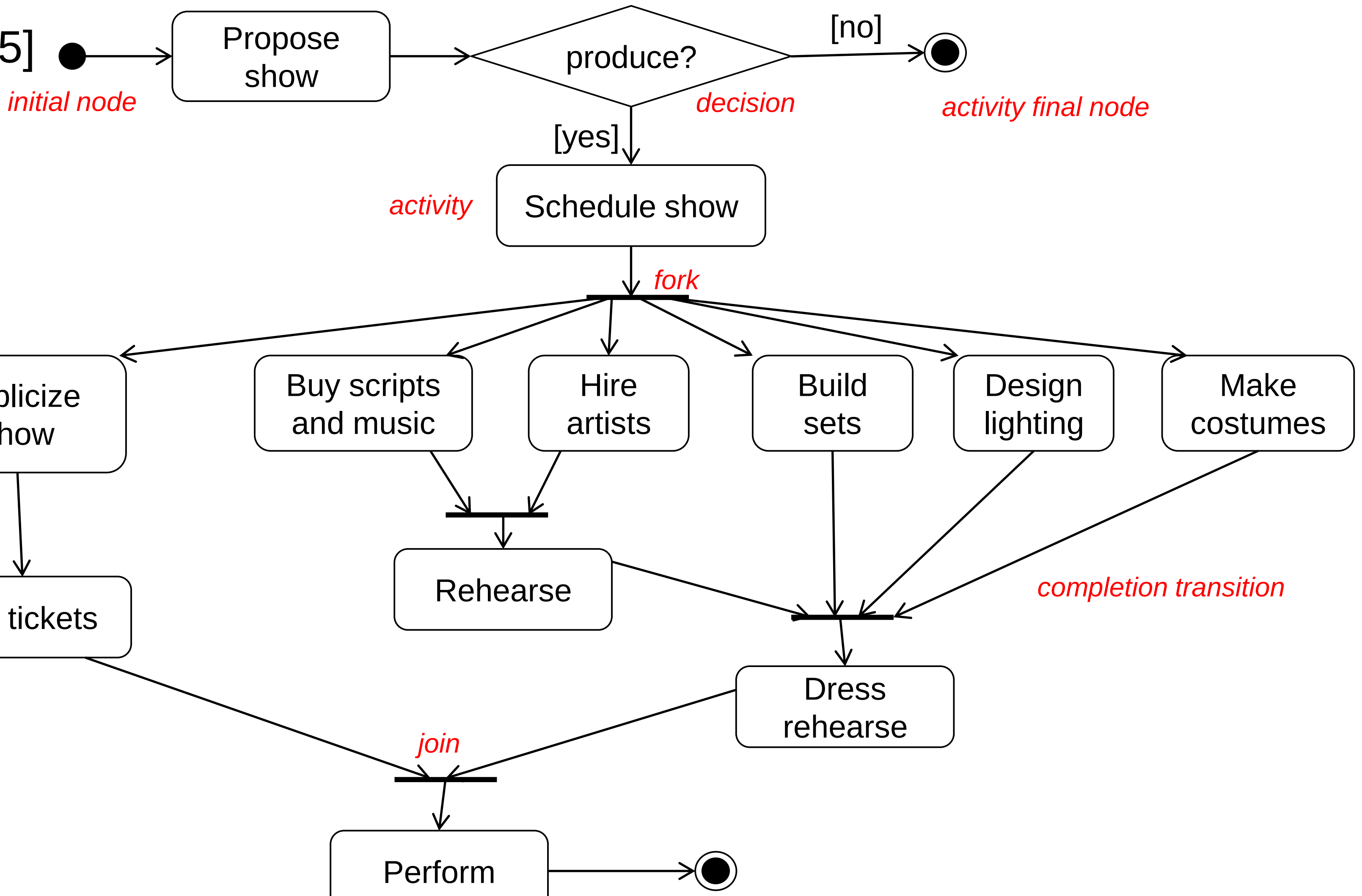
- the real-world workflows of a human organization.
- Software activities.

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.

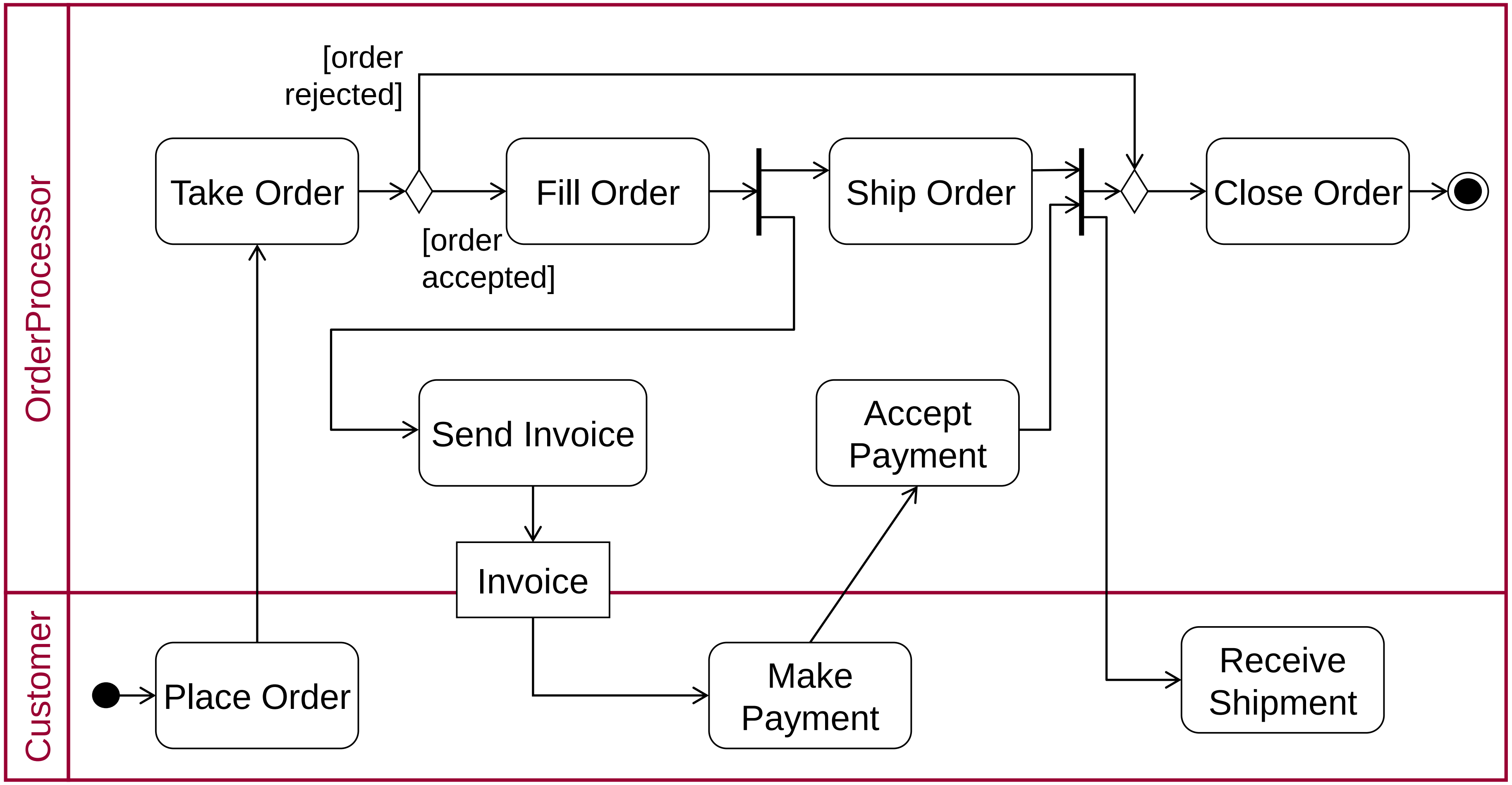
The input and output parameters of an activity can be shown using flow relationships connecting the action and object nodes.

Activity Diagram

[RJB05]



Activity Diagram with Partitions



Interaction View

[RJB05]

The interaction view describes sequences of message exchanges among the parts of a system.

This view shows the flow of control across many objects.

An interaction is based on a structured classifier or a collaboration.

A role is a slot that may be filled by objects in a particular use of an interaction.

The interaction view is displayed in two diagrams focused on different aspects:

- sequence diagrams
- communication diagrams

Sequence Diagram (1/4)

[RJB05]

A sequence diagram shows a set of messages arranged in time sequence.

- Each role is shown as a lifeline – a vertical line that represents the role over time through the entire interaction.
- Messages are shown as arrows between lifelines.

Structured control constructs such as loops, conditionals, and parallel execution are shown as nested rectangles with keywords and one or more regions.

Sequence Diagram (2/4)

[RJB05]

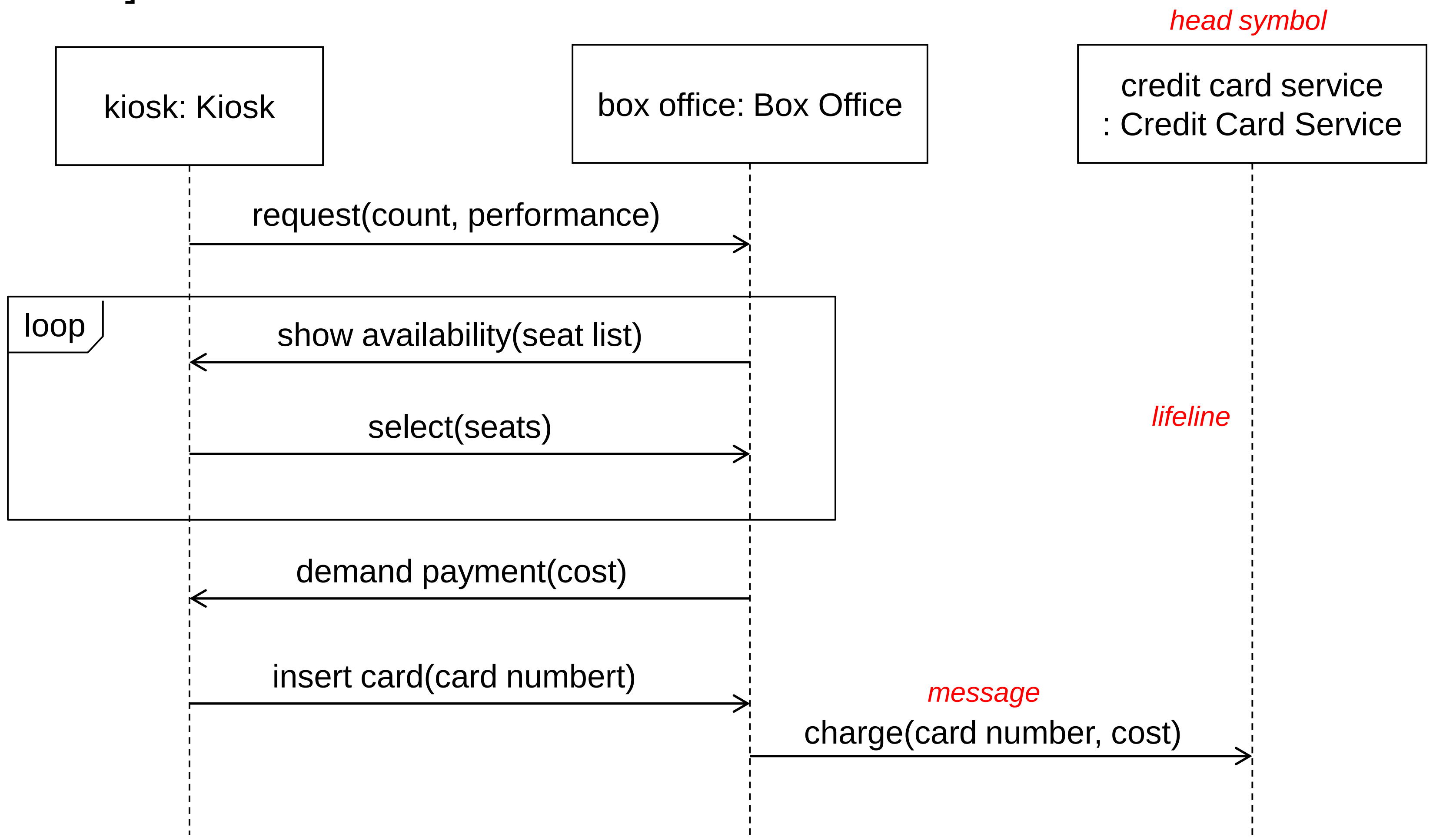
A sequence diagram can show a scenario – an individual history of a transaction.

A sequence diagram is often used to show the behavior sequence of a use case.

When the behavior is implemented, each message on a sequence diagram corresponds to an operation of a class or an event trigger on a transition in a state machine.

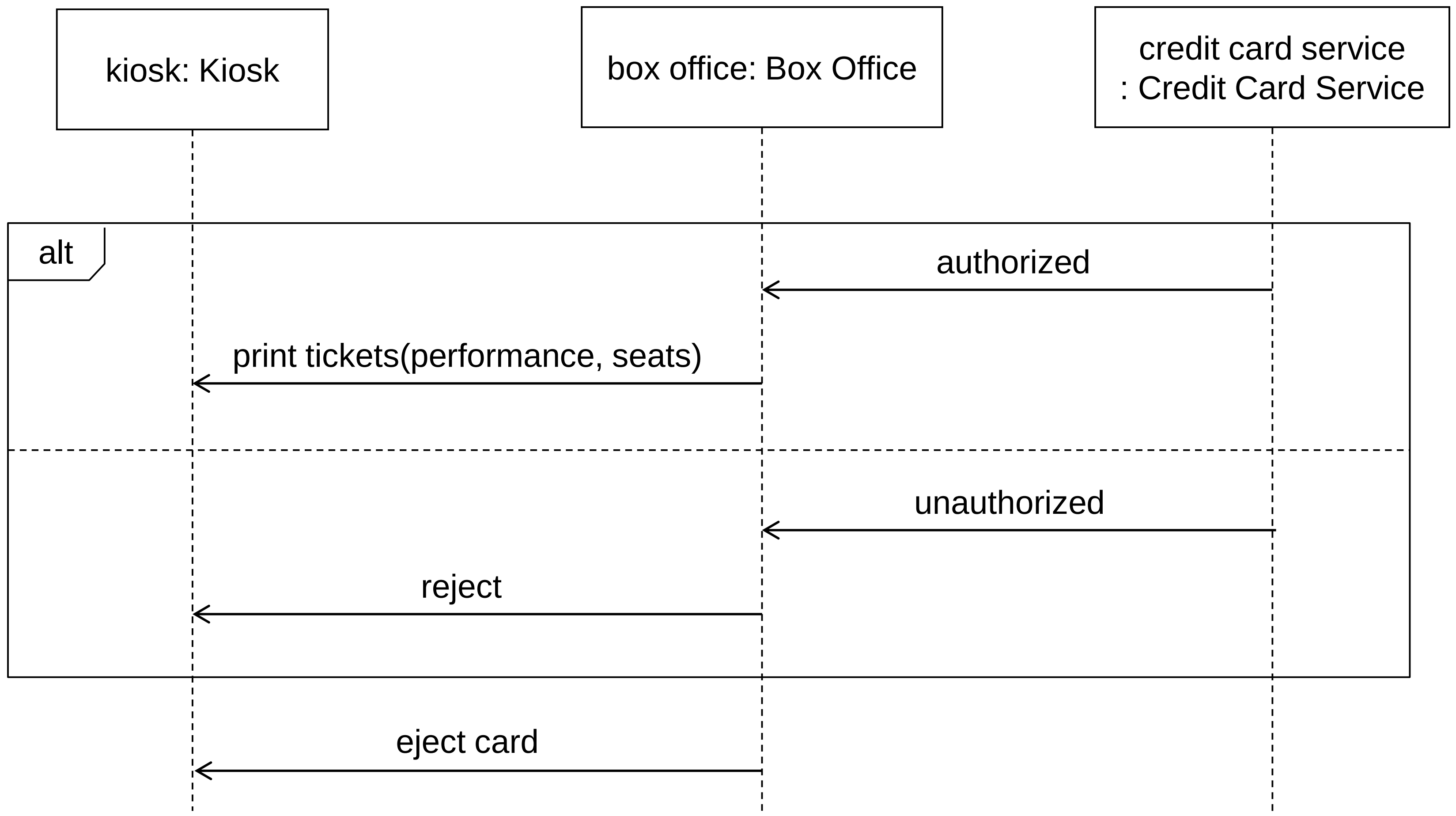
Sequence Diagram (3/4)

[RJB05]


head symbol
kiosk: Kiosk
box office: Box Office
credit card service : Credit Card Service
request(count, performance)
loop
show availability(seat list)
select(seats)
lifeline
demand payment(cost)
insert card(card numbert)
message
charge(card number, cost)

Sequence Diagram (4/4)

[RJB05]



Communication Diagram (1/3)

[RJB05]

A communication diagram shows roles in an interaction as a geometric arrangement.

- Each rectangle shows a role – a lifeline representing the life of an object over time.
- The messages among objects playing roles are shown as arrows attached to connectors.

The sequence of messages is indicated by sequence numbers prepended to message descriptions.

Communication Diagram (2/3)

[RJB05]

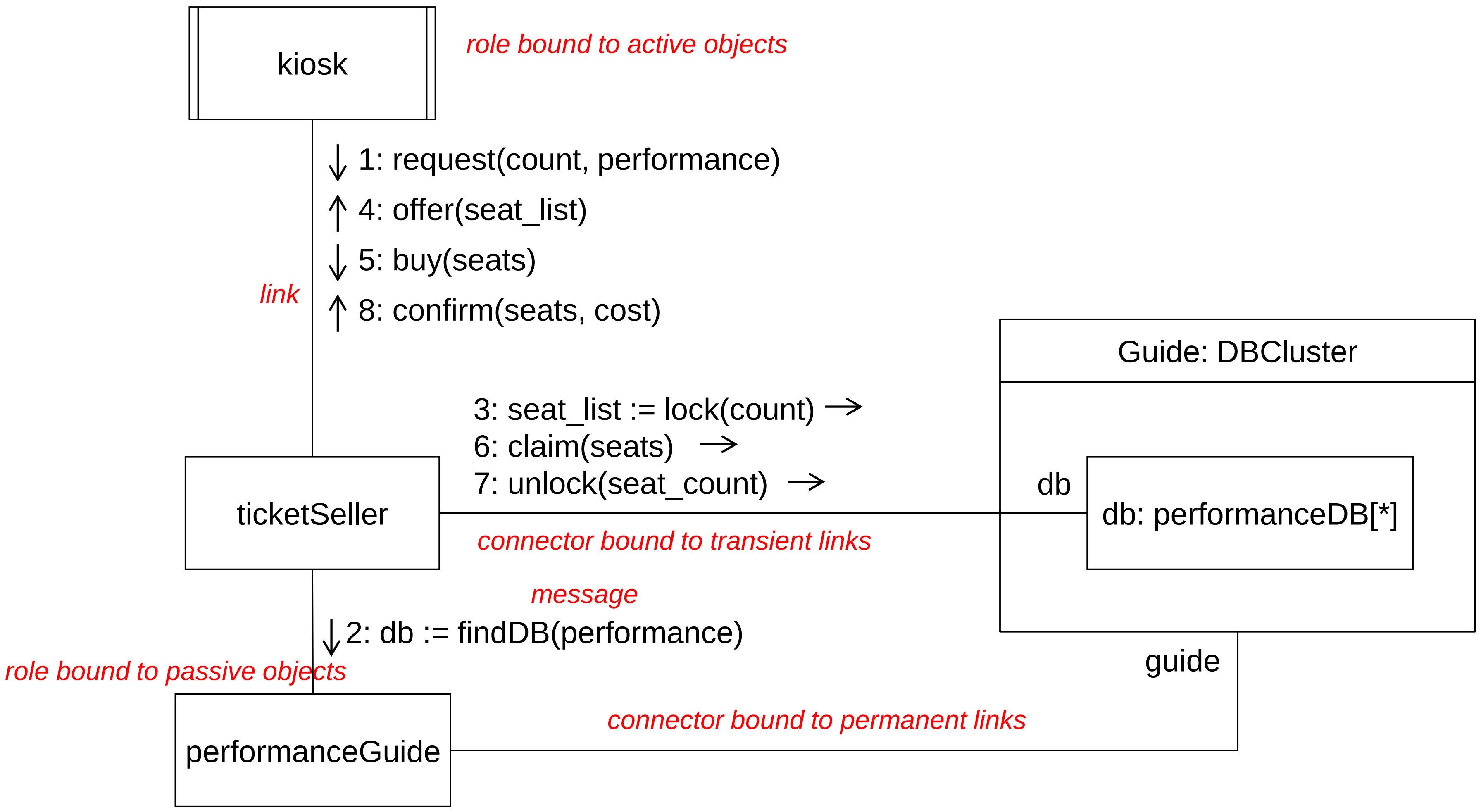
One use of a communication diagram is to show the implementation of an operation.

A collaboration shows the parameters and local variables of the operation as roles, as well as more permanent associations.

When the behavior is implemented, the message sequencing on a communication diagram corresponds to the nested calling structure and signal passing of the program.

Communication Diagram (3/3)

[RJB05]


kiosk
role bound to active objects
↓ 1: request(count, performance)
↑ 4: offer(seat_list)
↓ 5: buy(seats)
↑ 8: confirm(seats, cost)
link
Guide: DBCluster
3: seat_list := lock(count) →
6: claim(seats) →
7: unlock(seat_count) →
db
ticketSeller
db: performanceDB[*]
connector bound to transient links
message
↓2: db := findDB(performance)
role bound to passive objects
guide
connector bound to permanent links
performanceGuide

Sequence Diagrams and Communication Diagrams

[RJB05]

Both sequence diagrams and communication 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 communication diagram shows the relationships among roles geometrically and relates messages to the connectors, but time sequence is less clear.

Each diagram should be used when its main aspect is the focus of attention.

Deployment View (1/2)

[RJB05]

A deployment diagram represents the deployment of run-time artifacts on nodes.

- An artifact is a physical implementation unit such as a file.
- A node is a run-time resource such as a computer, device, or memory.

An artifact may be a manifestation (implementation) of one or more components.

This view permits the consequences of distribution and resource allocation to be assessed.

Deployment View (2/2)

[RJB05]

Two different levels of deployment can be depicted using deployment diagrams: a descriptor level and an instance level.

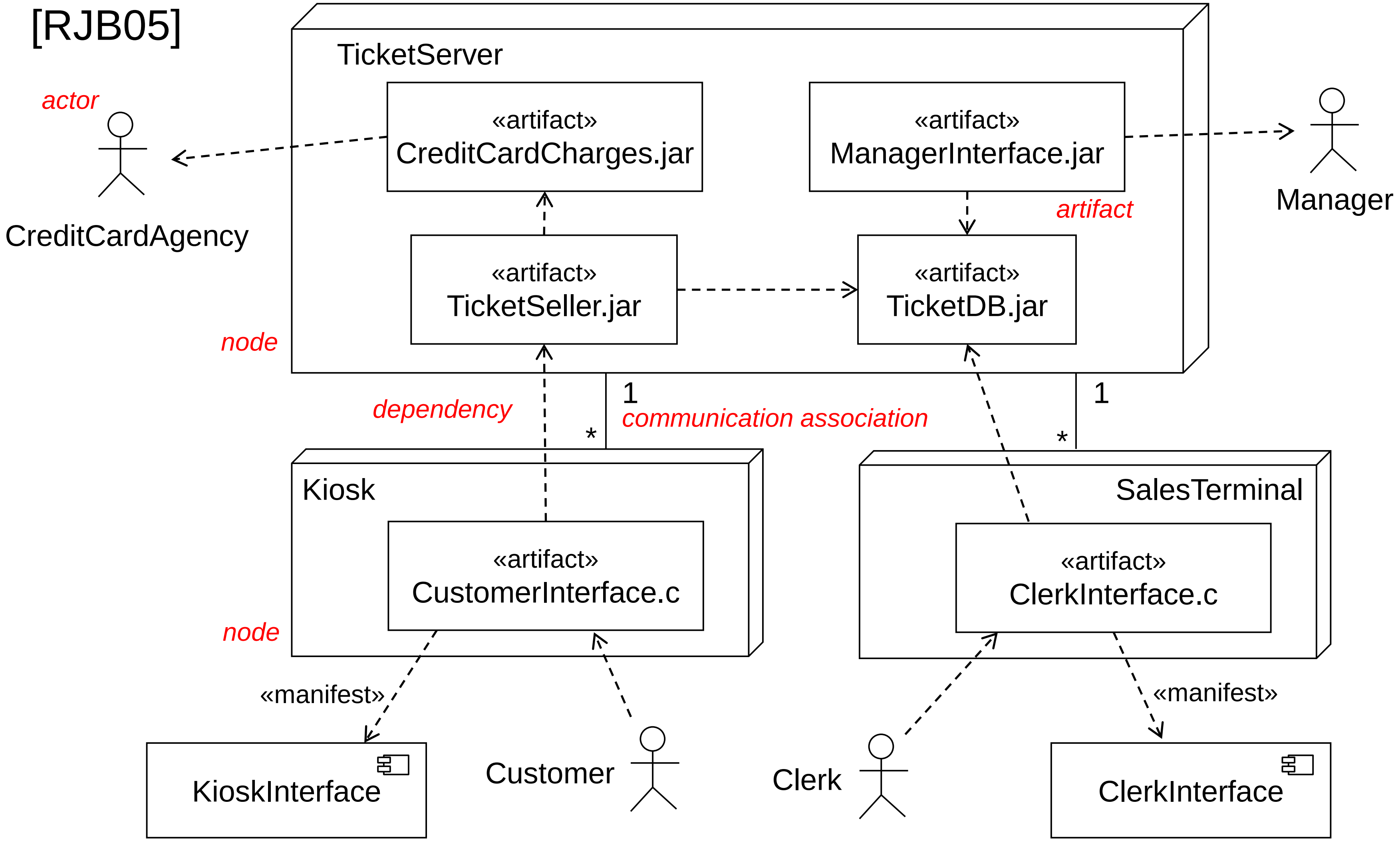
- Descriptor-level deployment diagrams show the kinds of nodes in the system and the kinds of artifacts they hold.
- Instance-level diagrams show the individual nodes and their links in a particular configuration of the system.

In a descriptor-level diagram, an artifact type can be located on different kinds of nodes, and different artifact types can manifest the same kind of component.

In modeling the deployment of a system in different levels, the instance-level information must be consistent with the descriptor-level information

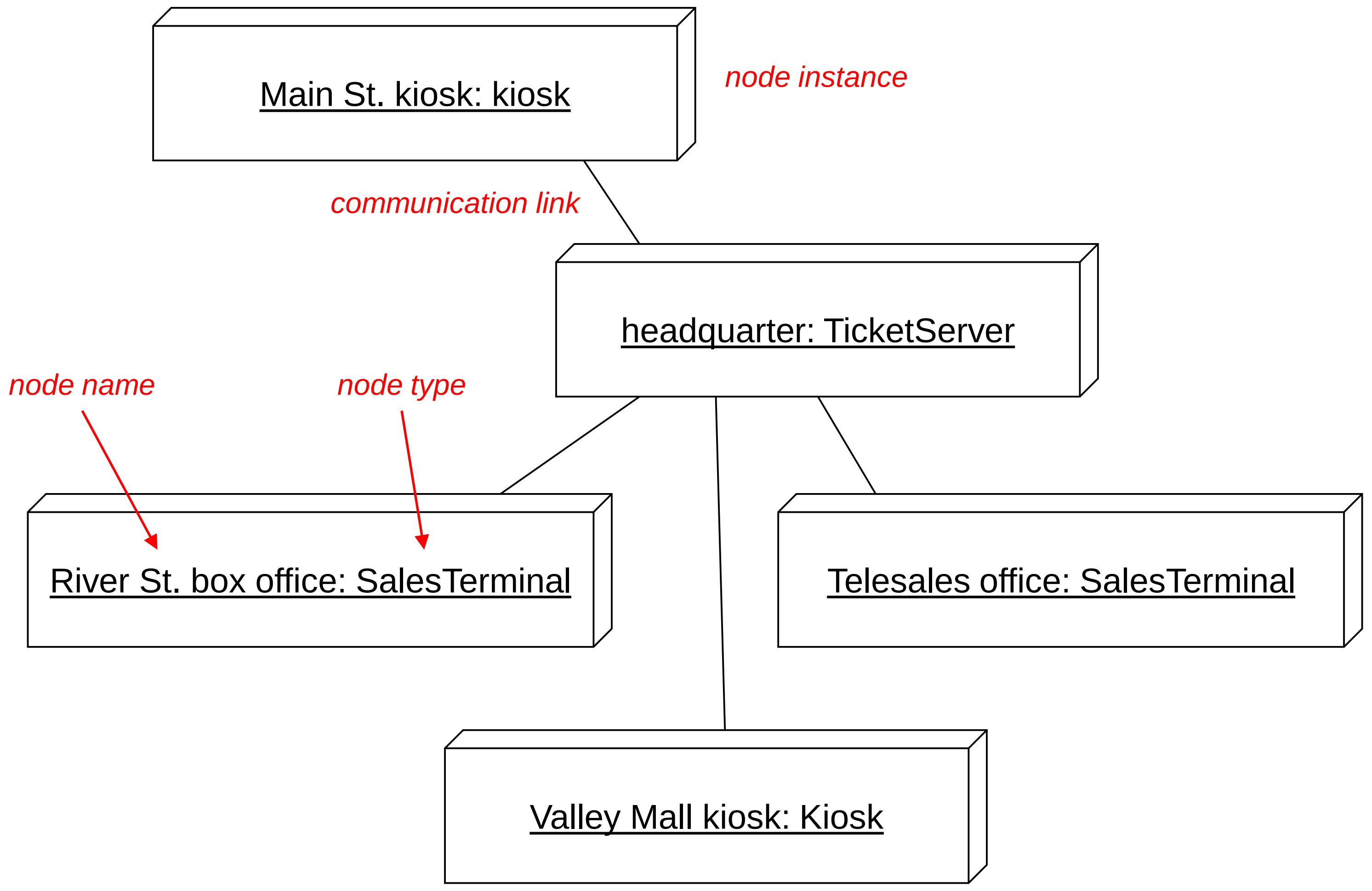
Deployment Diagram (Descriptor Level)

[RJB05]



Deployment Diagram (Instance Level)

[RJB05]



Model Management View (1/3)

[RJB05]

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.
- 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.

Model Management View (2/3)

[RJB05]

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.

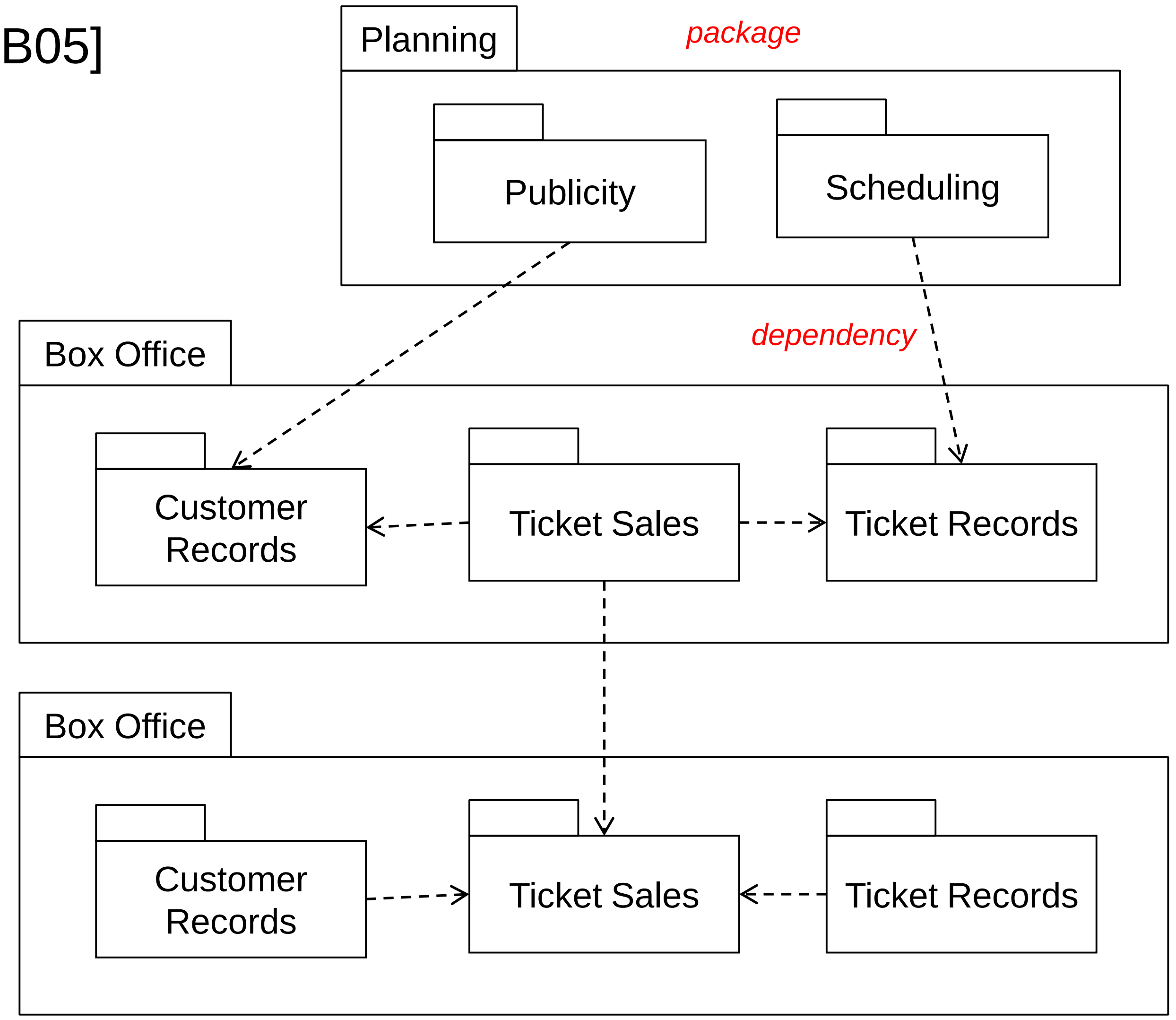
For example, an analysis model and a design model.

A model may be shown as a special kind of packages, but usually it is sufficient to show only the (ordinary) packages.

Model management information is usually shown on package diagrams, which are a variety of class diagram.

Model Management View (3/3)

[RJB05]



References

[RJB05] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual, 2/e*, Addison-Wesley, 2005

[OMG11] OMG, *Unified Modeling Language (OMG UML) Superstructure, Version 2.4.1*, OMG Document: formal/2011-08-06, 2011