08 Use Cases & UML Overview

- OMG Tutorial Series
  - [cK00] Kobryn Cris, "Lecture 1: Introduction to UML: Structural and Use Case Modeling Object Modeling with OMG," UML Tutorial Series
    http://www-edlab.cs.umass.edu/cs520/OMG-Tutorials/Tut1IntroToUML.ppt
  - [OSBB00] Övergaard, Gunnar, Bran Selic, Conrad Bock and Morgan Björkander, "Lecture 2: Behavioral Modeling with UML," Object Modeling with OMG UML Tutorial Series
    http://www-edlab.cs.umass.edu/cs520/OMG-Tutorials/Tut2BehaviorModeling.ppt
  - [PSWD00] Palmkvist, Karin, Bran Selic, Jos Warmer and Nathan Dykman, "Lecture 3: Advanced Modeling with UML," Object Modeling with OMG UML Tutorial Series
    http://www-edlab.cs.umass.edu/cs520/OMG-Tutorials/Tut3AdvModeling.ppt
  - Note: This version of the tutorial series is based on OMG UML Specification v. 1.4, UML Revision Task Force recommended final draft, OMG doc# ad/01-02-13.
- Other
  - [BJR98] **The Unified Modeling Language User Guide** by Grady Booch, Ivar Jacobson, James Rumbaugh Addison-Wesley Pub Co; 1st edition (1998)
  - [dB03] Bell, Donald, "UML basics: An introduction to the Unified Modeling Language," The Rational Edge, June 2003
    http://www-106.ibm.com/developerworks/rational/library/769.html

Use-case modeling strategy

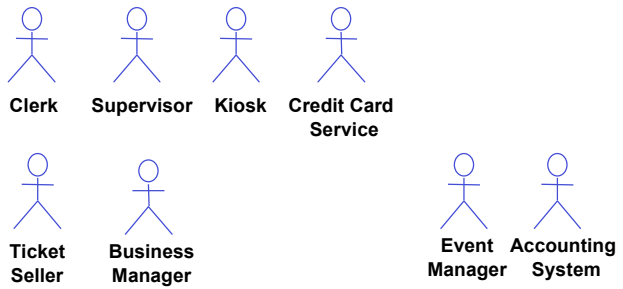
- Where to start?
  - with the human and other (external) systems that will use/interact with system to be developed ⇒ **Actors**
- How to determine what the system should do?
  - for each Actor, list possible scenarios ⇒ **Candidate Use Cases**
- How to manage a large number of use cases?
  - identify subset to emphasize for guiding design ⇒ **Focal Use Cases**
- How to know when to stop?
  - diagram actor/use-case relationships ⇒ **Use Case Diagram**
- How to describe the use cases?
  - Describe all/essential at high/detailed level? ⇒ **Use Case Descriptions**
- How to check that use cases are correct?
  - play out each use case before an audience of the stakeholders ⇒ **Use Case Roleplay**

This and following few slides are adapted from: Biddle, Robert, James Noble, Ewan Tempero "Patterns for Essential Use Cases"

Actors

- Design a program for a booking office of an arts center. There are several theatres, people may reserve seats at any theatre for any future event, and people may subscribe to a series of events. People need to be able to discuss seat availability, where seats are located, and how much they cost. When people make a choice, the program should print the price, record the selection, and print out a ticket
- Actors?
  - People (above) = buyer? surrogate for buyer (clerk, kiosk, on-line, agency)?
  - Other people = event manager, business manager. … ?
  - Systems = credit card system, banking system, accounting system, …??

Box Office Actors



consider only **direct** actors

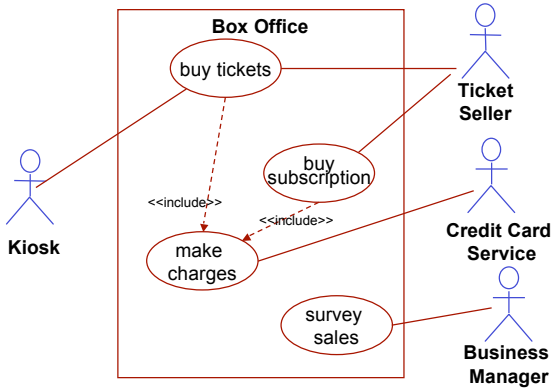
Candidate Use Cases

- Ticket Seller:
  - Find event (dates, times), determine seat availability, determine seat location, determine seat price, reserve and issue ticket, ditto for subscriptions?
- Kiosk:
  - Tickets only
- Event Manager:
  - Add event, schedule performance, modify performance information, other?
- Business Manager:
  - Print report
- Accountant:
  - Print sales information
- Credit Card Service
  - Authorize charges

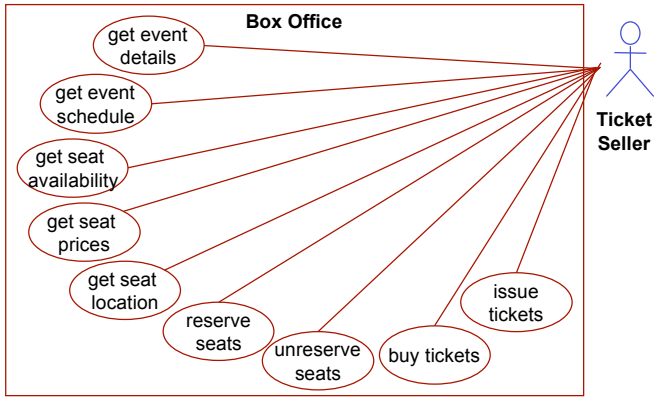
"Focal" Use Cases

- Issues
  - Even small systems can have a large number of use cases (~50 for a small system; ~100's for a medium system)
  - Some central, some not
  - Some will be easy to implement, some difficult
  - Different stakeholders will value the use cases differently
- Strategy
  - Ranking
  - Other elicitation techniques
- Example -- **Ticket Seller**:
  - List event performances, report event details, report availability of seats, show location of seats, **buy tickets (subscriptions), process payments**

Box Office Use Case Diagram



Box Office Use Case Diagram



Questions to ask

- Is there an actor representing every kind of user who will use the system?
- Is there a system actor for every (legacy) system with which this system needs to communicate?
- Can each actor do everything they need to do using only the use cases they are related to?
- Are any obvious use cases missing?
  - use case models are often symmetric: if there are use cases for creating bookings, printing booking receipts, printing performance receipts, and canceling performances, perhaps there should also be use cases for canceling bookings and creating performances.
- Unless you are on a small system (if you have not more than 15-20 use cases) draw one use case diagram for each actor (or for a few related actors), rather than one diagram for the whole system.

CRUD analysis

- CRUD = Create, Read, Update, or Delete
- Identify CRUD classes and check whether any of the other CRUD use cases are needed.
- Consider the rest of the classes in the domain model and check if they should also have CRUD use cases.
  - Rename these uses cases where appropriate
- Example: **Get Event Details** ⇒ **Read Event**
  - we might want: Create Event, Delete Event, and Update Event
- Not all of the CRUD use cases are needed for every concept
  - so CRUD analysis should not be applied blindly.
  - not all classes in the domain model should have CRUD analysis applied to them.

Reporting Use Cases

- Reporting is very important for lots of systems, but
  - boring for implementers, so they can underestimate the effort required to produce reports.
  - boring to model
  - boring to one of users or stakeholders, while simultaneously crucial to the others.
- If it's important to someone, you have to model it
- Will also ensure you capture all the important cases
- You should have **at least twenty (?)** reporting use cases.
  - Occupancy Report, Report Monthly Sales, Report Seat Availability, Occupancy Report by Theatre, Occupancy Report by Event, Occupancy Report by Performance, Occupancy Report by Week, … are all of interest to the Business Manager, but may be useful to the Ticket Seller when answering questions of the form "Are the plenty of free seats for tomorrow's performance of Dracula?".

Use Case Descriptions

Create JSP pages with Struts taglibs
Revisions

Date	Revision	Description	Author
1/21/2003	0.9	First Draft	Kenneth Lewelling
4/4/2003	0.93	Converted document to xml for use with Maven.	Kenneth Lewelling

Characteristic Information
Goal in Context: Create a JSP page in OpenCms that uses Struts tag libraries.
Preconditions: A JSP page exists in the VFS, the user has read-write rights to it, and the user has the file locked.
Successful End Condition: A JSP page is edited, a reference to a Struts tag library has been made and the JSP page successfuly loads.
Failed End Condition: The JSP page can not be saved, or it can be saved but does not successfuly load, or the JSP page remains unchanged.
Primary Actor: Web Developer
Trigger: The actor selects edit sourcecode on the context menu of the JSP file.
Success Scenario
1. The actor enters a Struts taglib definition such as `" < %@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" % > "` at the top of the file.
2. The actor uses one of the tags in the library specified in the JSP page such as: `< bean:message key="index.heading"/ >` where index.heading is defined in the message resource.
3. The actor saves the JSP page.
4. The actor views the JSP page and it succesfuly renders.

Extensions
3a. The actor cancels the changes. The JSP page remains unchanged.
4a. The page doesn't render properly
4aa. The taglib definition is misspelled, or the tag used is misspelled or invalid.
4ab. The mistake is corrected by the actor.
4ac. Continue with step 3.
Sub-Variations
There are no sub-variations.
Related Information
Super Ordinate Use Case:
Subordinate Use Case:
Open Issues
None.

RUP-style Use Case Descriptions

Other Approaches

- Constantine & Lockwood
  - Identify "essential" use cases
  - *An essential use case is a structured narrative, expressed in the language of the application domain and of users, comprising a simplified, generalized, abstract, **technology-free and implementation independent** description of one task or interaction that is complete, meaningful, and well-defined from the point of view of users in some role or roles in relation to a system and that embodies the purpose or intentions underlying the interaction.*

gettingCash

User Action	*System Response*
insert card	
	read magnetic stripe
	request PIN
enter PIN	
	verify PIN
	display transaction menu
press key	
	display account menu
press key	
	prompt for amount
enter amount	
	display amount
…	
	…

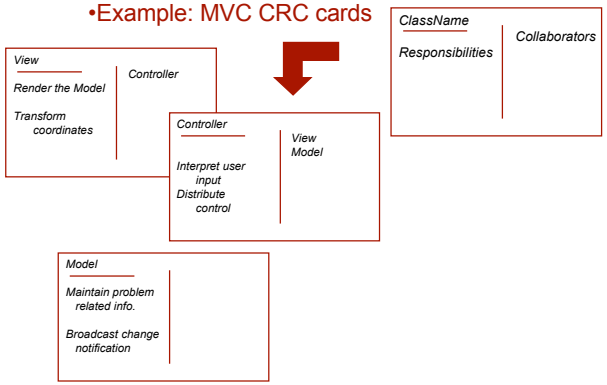
⇒

gettingCash

User Intention	*System Responsibility*
identify self	
	verify identity
	offer choices
choose	
	dispense cash
take cash	

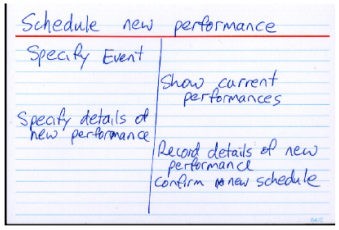
CRC Cards

- CRC = class name, responsibilities, and collaborators
  - Example: MVC CRC cards



Other Approaches

- Biddle, Noble, Tempro
  - Use Case cards -- similar to Beck, et al CRC cards
  - Simpler, less implementation dependent, saves "unnecessary" documentation



Use Case Roleplay

- Example: Get Seat Availability
  - The ticket seller ("user") is using the "system" to determine whether the seats requested by the customer for a performance are in fact available.
- Take 1:
  - **User:** I say which performance I want and the system shows me the performance details.
    CUT! — it's the system's job to say what the system does. This is often just an error made by the role-player, but can also indicate confusion as to where the system boundary is.
- Take 2:
  - **User:** I say which performance I want.
  - **System:** I display the performance details and say whether or not the seats are available.
    CUT! — the seats haven't been specified yet.

Use Case Roleplay

- Example: Get Seat Availability

  The ticket seller ("user") is using the "system" to determine whether the seats requested by the customer for a performance are in fact available.
- Take 3
  - **User:** I say which performance I want. User pauses waiting for a response, then Looks over to the person playing the system, who is still looking at the use case card, and doesn't realize he's being cued. System?
  - **System:** You're supposed to say what seats you want to know about too. Points at card.
  - **User:** Oh, right

    CUT. The roleplay does not allow anyone to hide — all participants have to engage with what the use case is about.
- Take 4:
  - And so on. . .

Roleplay

- Using roleplay to assist use case checking is not strictly necessary, but it does harness several human skills:
  - abilities of the people playing the roles to identify with the roles
  - focus more intently on the user intention or the system responsibility, and to detect problems.
- But, it is another checking practice that is subject to diminishing returns:
  - the whole process can be annoying and time consuming
  - many people object to performing in front of the rest of the team and group activities can also be soured by managerial involvement

Use Case Dialog Patterns

- Alarm Use Case
  - How do you have the system inform the user about something?
  - Write a use case that begins with the system taking the responsibility to warn the user.
  - If the alarm is important, you may need to include a Confirming Step
- Requesting Use Case
  - How do you write a use case when the user needs to know something from the system?
  - Write a use case where the actor describes the information they require, and then the system presents that information.
- Monitoring Use Case
  - How do you write a use case where the user often needs to know about a relatively small amount of important information from the system.
  - Write a use case where the system presents that information.

Use Case Dialog Patterns

- Commanding Use Case
  - How do you have the user get the system to do something?
  - Write a use case where the user provides information on the request, and the system has the responsibility for performing the command
- Prompting Step
  - How should you write a use case when the system knows some information that would help the use make a decision?
  - Give the system the responsibility of offering that information before the user makesthe decision.
- Confirming Step
  - How should you write a use case when it is important that correct information is communicated between the actor and the system?
  - Require the actor or system to confirm the information.

Use Case Dialog Patterns

Alarm Use Case Examples

Warn of start of performance

User Intention	System Responsibility
	Signal "performance about to start"
	Show name, theater, and times of performance

Warn theater performance undersold

User Intention	System Responsibility
	Signal "performance undersold"
	Show name, theater, time or performance, and percentage of seats sold
Confirm warning	

Confirming Step Example

Pay for reservation

User Intention	System Responsibility
	Present reservation details
	Offer payment methods
Choose payment method	
Supply payment details	
Confirm method and details	
	Accept payment
	Confirm booking

Requesting Use Case Example

Get Seat Prices

User Intention	System Responsibility
	Offer performances
Choose performance	
	Show prices for chosen performance

Commanding Use Case Example

Print Performance Schedule

User Intention	System Responsibility
Chose start and end dates	
	Print schedule of performances from start to end date

Monitoring Use Case Example

Show Today's Performances

User Intention	System Responsibility
	Show today's performances

Prompting Step Example

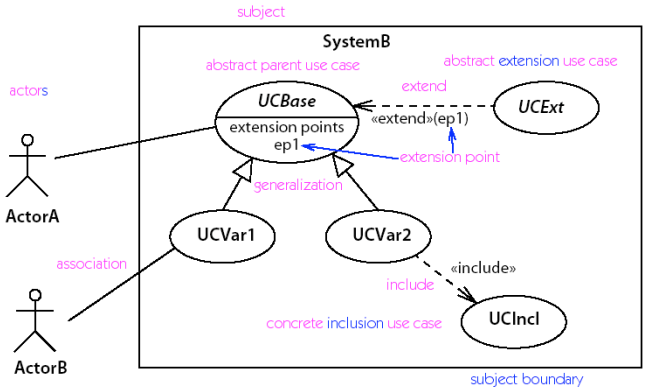
Reserve seats for performance

User Intention	System Responsibility
	Offer unreserved seats
Choose seats	

Organizing Use Cases

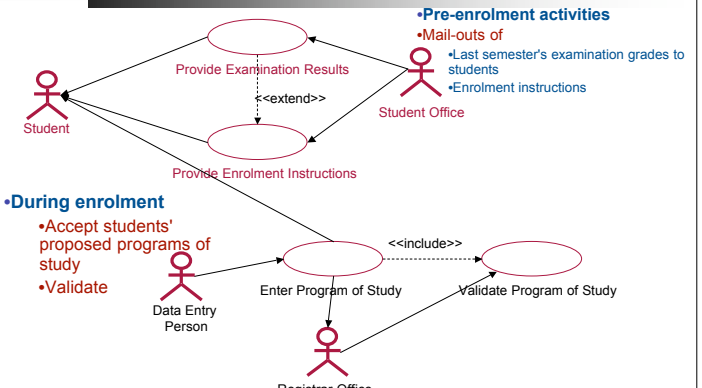
- How do you model errors and exceptions in use cases?
  - Use **extending** use cases.
- How do remove commonality between use cases?
  - Make a new use case containing the common steps, and **include** it in the use cases that have the common steps.
- How do you handle more general and more specific use cases that do the same kind of thing?
  - Use **specialization**
  - Prefer inclusion to specialization.
- How do you model use cases than can only operate under certain circumstances?
  - Use **pre- and post-conditions** to control when use-cases are permissible.
  - Prefer extensions to conditions.
  - Pre- and post-conditions should match in a complete model.

Use Case Notation



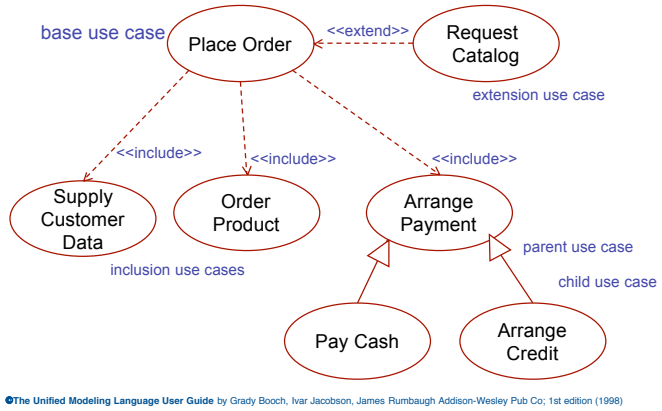
The Unified Modeling Language User Guide by Grady Booch, Ivar Jacobson, James Rumbaugh Addison-Wesley Pub Co; 1st edition (1998)

Enrollment Example



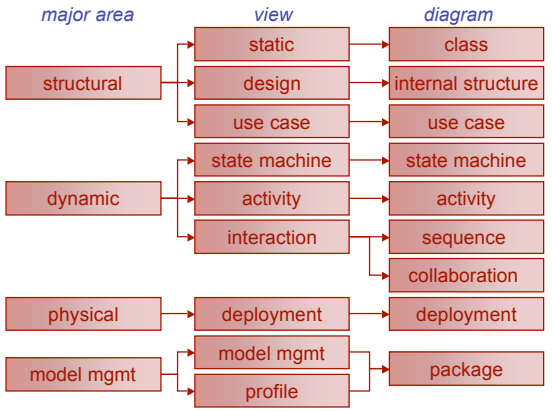
© MACIASZEK, L.A. (2001): Requirements Analysis and System Design. Developing Information Systems with UML, Addison Wesley Copyright © 2000 by Addison Wesley

Another Example

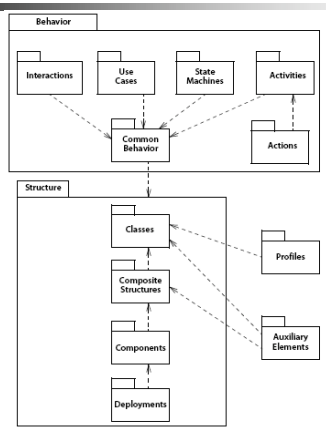


©The Unified Modeling Language User Guide by Grady Booch, Ivar Jacobson, James Rumbaugh Addison-Wesley Pub Co; 1st edition (1998)

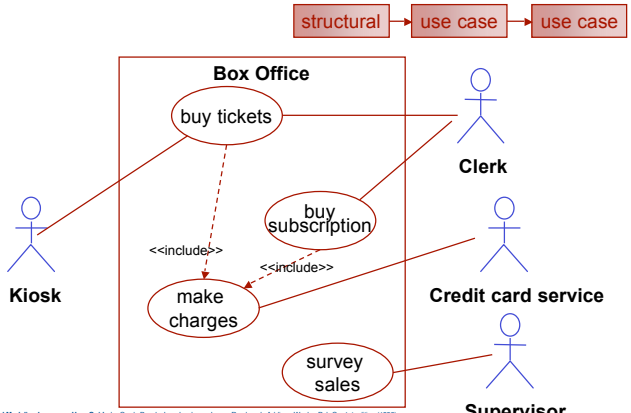
UML 2.0



Structure & Behavior



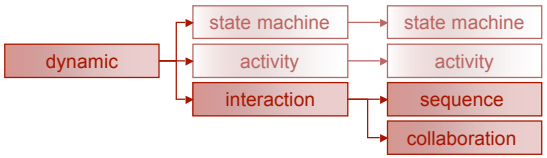
Use Case Diagram



©The Unified Modeling Language User Guide by Grady Booch, Ivar Jacobson, James Rumbaugh Addison-Wesley Pub Co; 1st edition (1998)

Use Case Realizations

- The use case diagram presents an outside view of the system
- Interaction diagrams describe how use cases are realized as interactions among societies of objects
- Two types of interaction diagrams
  - Sequence diagrams
  - Collaboration diagrams

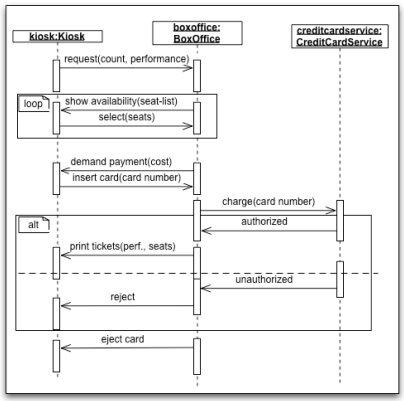


sequence diagram

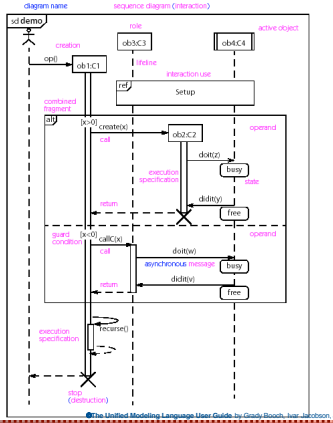
- an interaction diagram that details how operations are carried out
  - what messages are sent and when
  - are organized according to time
  - time progresses as you go down the page
  - objects involved in the operation are listed from left to right according to when they take part in the message sequence.

Symbol	Meaning
→	simple message which may be synchronous or asynchronous
- - →	simple message return (optional)
→	a synchronous message
⇒	an asynchronous message

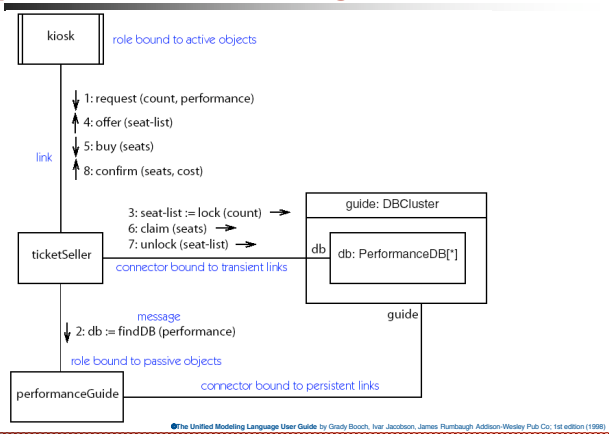
Sequence Diagrams



Sequence Diagram Notation



Communication Diagram



Classes

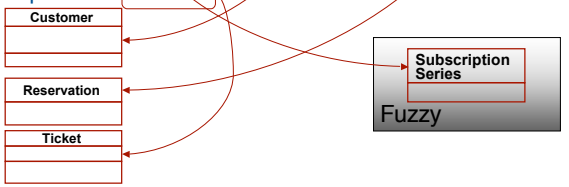
- A class is a collection of objects with common structure, common behavior, common relationships and common semantics
- Classes may be found by examining:
  - written requirements
  - objects in sequence and collaboration diagrams
- A class is drawn as a rectangle with three compartments
  - name
  - attributes
  - operations
- Classes should be named using the vocabulary of the domain
  - Naming standards should be created
  - e.g., all classes are singular nouns starting with a capital letter

Class Modeling

- Captures **system state** – the function of the system's information content at a point in time
- Class modeling and use case modeling are typically conducted in parallel
- A **class diagram** shows the existence of classes and their relationships in the logical view of a system; in UML class diagrams elements include
  - Classes and their structure and behavior
  - Association, aggregation, generalization, dependency, and inheritance relationships
  - Multiplicity and navigation indicators
  - Role names

Find Classes from requirements

- Design a program for a booking office of an arts center. There are several theatres, people may reserve seats at any theatre for any future event, and people may subscribe to a series of events. People need to be able to discuss seat availability, where seats are located, and how much they cost. When people make a choice, the program should print the price, record the selection, and print out a ticket



Attributes

- The structure of a class is represented by its attributes
- Attributes may be found by examining class definitions, the problem requirements, and by applying domain knowledge
- More requirements
  - Each customer offering has a name and phone number
  - Each show has a name
  - Each performance has a date and time
  - Ticket has availability

Customer	Show	Performance	Ticket
name: String phone: String	name: String	date: Date time: TimeOfDay	available: Boolean

Operations

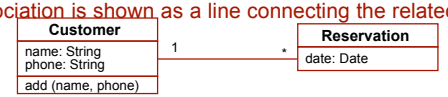
- The behavior of a class is represented by its operations
- Operations may be found by examining interaction diagrams

Relationships

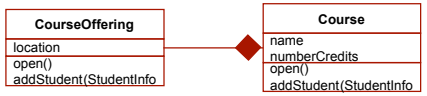
- Relationships provide a pathway for communication between objects
- Sequence and/or collaboration diagrams are examined to determine what links between objects need to exist to accomplish the behavior -- if two objects need to "talk" there must be a link between them
- Three types of relationships are:
  - Association
  - Aggregation
  - Dependency

Relationships

- An association is a bi-directional connection between classes
  - An association is shown as a line connecting the related classes

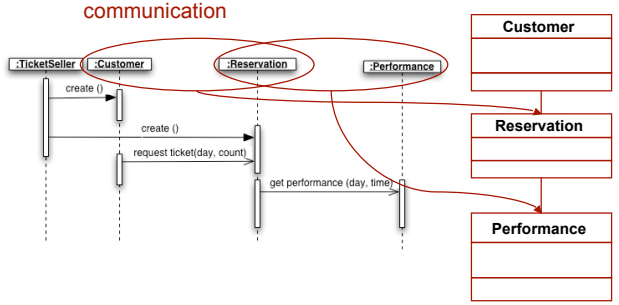


- An aggregation is a stronger form of relationship where the relationship is between a whole and its parts
- An aggregation is shown as a line connecting the related classes with a diamond next to the class representing the whole



Finding Relationships

- Relationships are discovered by examining interaction diagrams
  - If two objects must "talk" there must be a pathway for communication



Multiplicity and Navigation

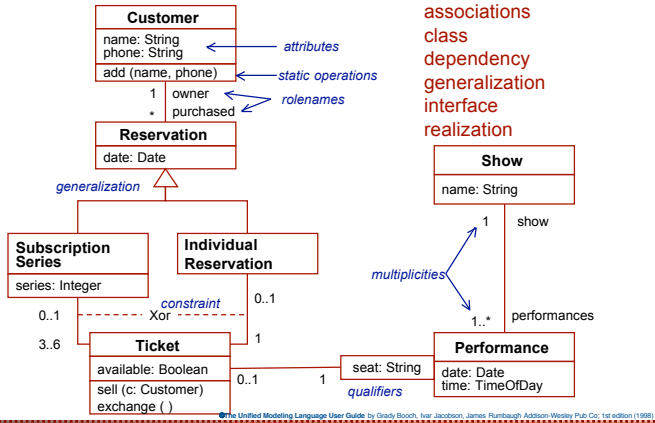
- Multiplicity defines how many objects participate in a relationships
  - Multiplicity is the number of instances of one class related to ONE instance of the other class
  - For each association and aggregation, there are two multiplicity decisions to make: one for each end of the relationship
- Although associations and aggregations are bi-directional by default, it is often desirable to restrict navigation to one direction
- If navigation is restricted, an arrowhead is added to indicate the direction of the navigation

Inheritance

- Inheritance is a relationships between a superclass and its subclasses
- There are two ways to find inheritance:
  - Generalization
  - Specialization
- Common attributes, operations, and/or relationships are shown at the highest applicable level in the hierarchy

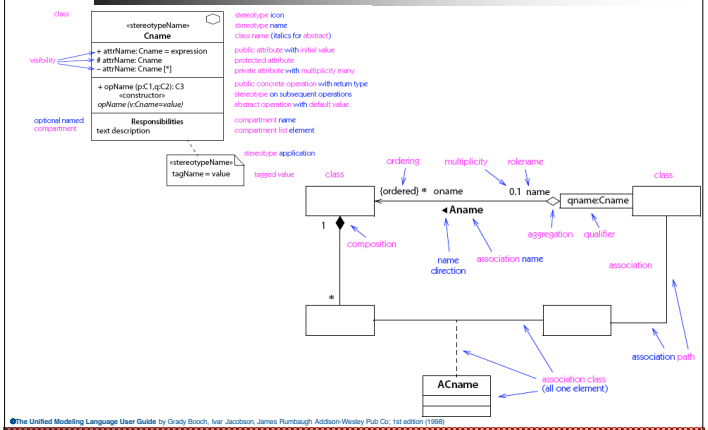
Class Diagram

structural → static → class



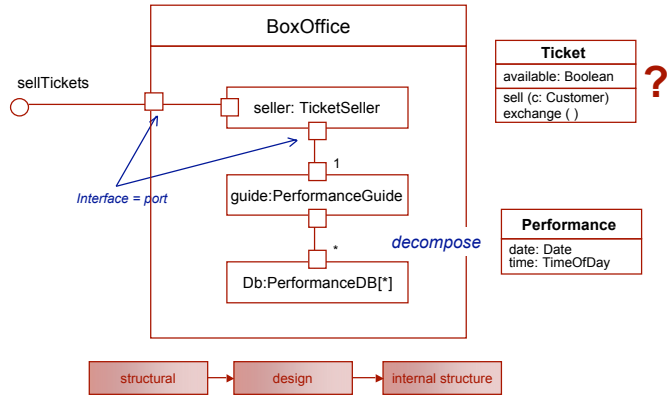
The Unified Modeling Language User Guide by Grady Booch, Ivar Jacobson, James Rumbaugh Addison-Wesley Pub Co; 1st edition (1998)

Class & associations -- notation



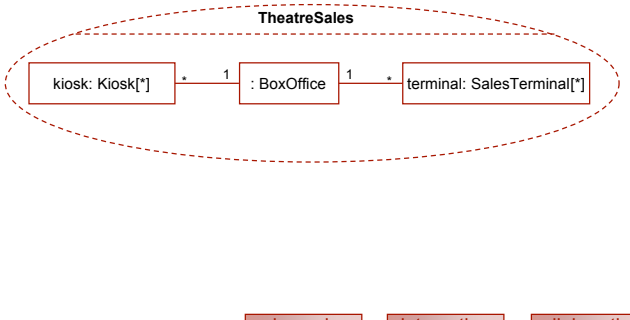
The Unified Modeling Language User Guide by Grady Booch, Ivar Jacobson, James Rumbaugh Addison-Wesley Pub Co; 1st edition (1998)

Internal Structure Diagram



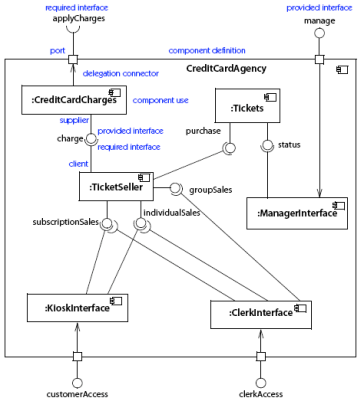
The Unified Modeling Language User Guide by Grady Booch, Ivar Jacobson, James Rumbaugh Addison-Wesley Pub Co; 1st edition (1998)

Collaboration Diagram



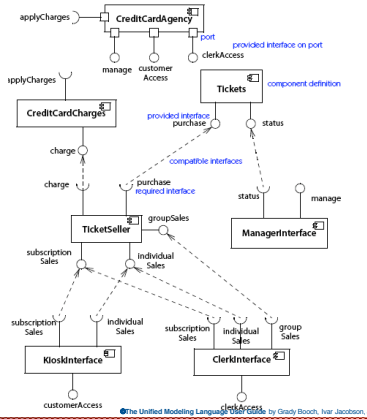
The Unified Modeling Language User Guide by Grady Booch, Ivar Jacobson, James Rumbaugh Addison-Wesley Pub Co; 1st edition (1998)

Component Definition



The Unified Modeling Language User Guide by Grady Booch, Ivar Jacobson, James Rumbaugh Addison-Wesley Pub Co; 1st edition (1998)

Component Diagram



©The Unified Modeling Language User Guide by Grady Booch, Ivar Jacobson, James Rumbaugh Addison-Wesley Pub Co; 1st edition (1998)

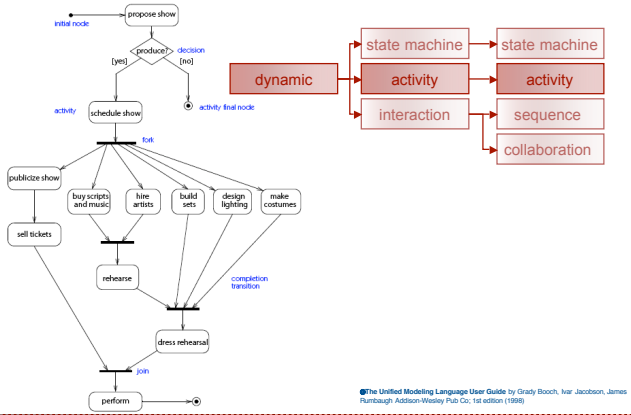
UML 2 Activity Diagrams

- object-oriented equivalent of flow charts and data flow diagrams (DFDs) from structured development
- typically used for
  - business process modeling
  - modeling the logic captured by a single use case or usage scenario
  - modeling the detailed logic of a business rule
- could potentially model
  - the internal logic of a complex operation
  - far better to simply rewrite the operation so that it is simple enough that you don't require an activity diagram

Activity Diagram Notation

- Initial node
  - filled circle = starting point of the diagram, not required
- Activity final node
  - filled circle with a border is the ending point, can have zero or more activity final nodes.
- Activity
  - rounded rectangles represent activities, may be physical or electronic
- Flow/edge
  - arrows on the diagram
- Fork
  - A black bar with one flow going into it and several leaving it, parallel activity.
- Join
  - A black bar with several flows entering it and one leaving it, denotes the end of parallel processing.
- Condition
  - a guard which must evaluate to true in order to traverse the node
- Decision
  - A diamond with one flow entering and several leaving
- Merge
  - A diamond with several flows entering and one leaving, all incoming flows must reach this point until processing continues, unless otherwise noted
- Partition
  - also called swimlanes, indicating who/what is performing the activities
- Sub-activity indicator
  - rake in the bottom corner of an activity indicates that the activity is described by a more finely detailed activity diagram.
- Flow final
  - circle with the X through it, process stops at this point.
- Use case
  - non-official? indication that an included use case is being invoked.

Activity Diagram



©The Unified Modeling Language User Guide by Grady Booch, Ivar Jacobson, James Rumbaugh Addison-Wesley Pub Co; 1st edition (1998)