UML Fundamental

NetFusion Tech. Co., Ltd.
Jack Lee

2008/4/7

OutLine

- Use-case diagram
- Class diagram
- Sequence diagram
- Communication diagram
- State machine
- Activity diagram

What is UML?

- **U**nified **M**odeling **L**anguage

  In short, the Unified Modeling Language (UML) provides industry standard mechanisms for visualizing, specifying, constructing, and documenting software systems.

3

Usable Model

- Accurate.
- Understandable.
- Consistent.
- Modifiable.

4

UML Features

- Views.
- Diagrams.
- Model elements.
- General mechanisms.
- Model Driven Architecture (MDA) features.

5

Features - Views

- Use-case view.
- Logical view.
- Implementation view.
- Process view.
- Deployment view.

6

Features - Diagrams

UML 2.0 has 13 types of diagrams, which can be categorized hierarchically as follows:

- Structure diagrams
- Behavior diagrams
- Interaction diagrams

7

Structure diagrams

Emphasize what things must be in the system being modeled.

- Class diagram
- Component diagram
- Composite structure diagram
- Deployment diagram
- Object diagram
- Package diagram

8

Behavior diagrams

Emphasize what must happen in the system being modeled.

- Activity diagram
- State Machine diagram
- Use case diagram

9

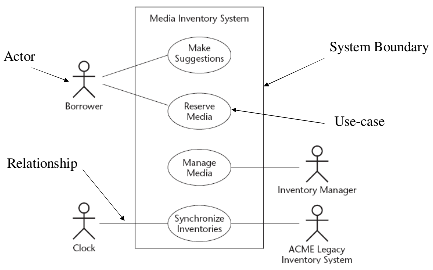
Interaction diagrams

A subset of behavior diagrams, emphasize the flow of control and data among the things in the system being modeled.

- Communication diagram
- Interaction overview diagram (UML 2.0)
- Sequence diagram
- UML Timing Diagram (UML 2.0)

10

Use-case diagram



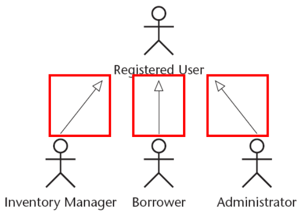
11

System & Actors

- System
  - The boundaries of the system, or subject, developed are defined. The system can itself be a classifier, or subject, that owns the set of use cases.
- Actors
  - someone or something that interacts with the system; it’s who or what uses the system.

12

Relationships between Actors



13

Use Cases

- A use case represents functionality for an actor. A use case in UML is defined as
  - “a set of actions performed by a system, which yields an observable result that is, typically, of value for one or more actors or other stakeholders of the system.”

14

Characteristics of a use case

- A use case is always initiated by an actor.
- A use case provides value to an actor.
- A use case is complete.

15

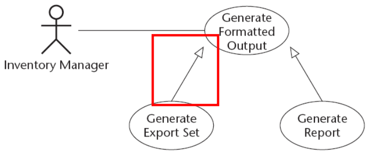
Relationships between Use Cases

There are three types of relationships between use cases:

- Generalization
- Extend
- Include

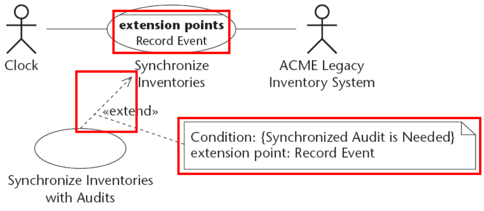
16

Generalization Relationship



17

Extend Relationship



Condition: {Synchronized Audit is Needed}
extension point: Record Event

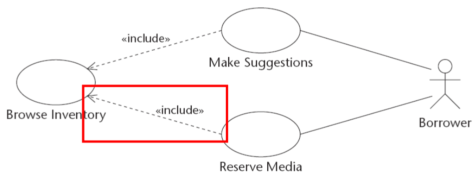
18

Extend Relationship (Cont.)

Don’t overdo it, but the extend relationship can come in handy in two distinct situations:

- When a system to be developed will potentially be deployed with varying sets of optional behavior
- When the deployment schedule of a system’s use cases requires deployment of functional, though not complete, use cases

Include Relationship

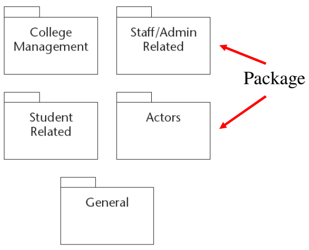


Include Relationship (Cont.)

- When a number of use cases have common behavior, this behavior can be modeled in a single use case that is included by the other use cases.
- When a use case includes another, the entire use case must be included.
- The idea of the include relationship is to model common behavior, such as logging information, that many use cases depend on, or "include."

21

Organizing Use Cases



22

Organizing Use Cases (Cont.)

- Use cases and actors can and often are placed into a substantial package structure.
- It is often useful to organize them along either functional lines or by use cases performed by specific actors.

23

Class Diagram

- A class diagram describes the static view of a system in terms of classes and relationships among the classes.

24

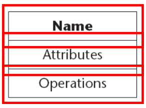
Class

A class is drawn with a rectangle, often divided into three compartments:

– The name compartment

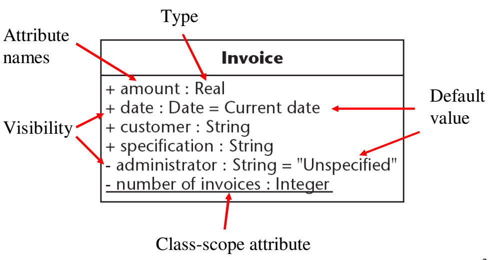
– The attributes compartment

– The operations compartment



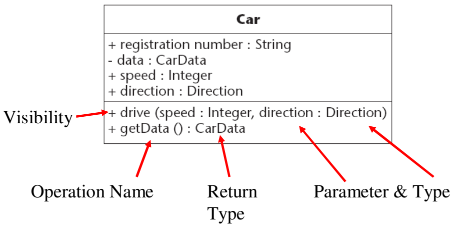
25

Attributes Compartment



26

Operations Compartment

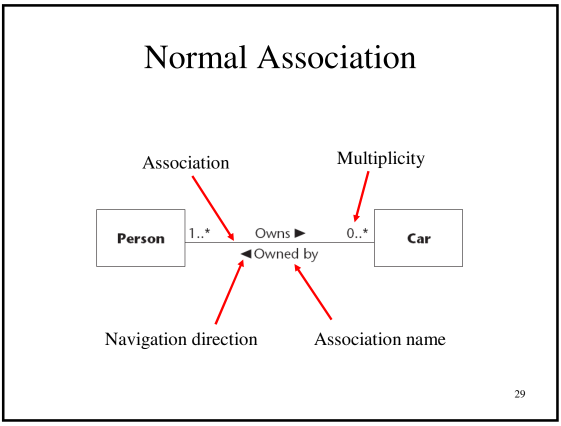


27

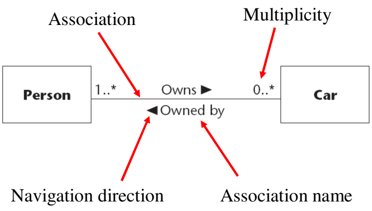
Class relationships

- Association
- Generalization
- Dependency
- Abstraction

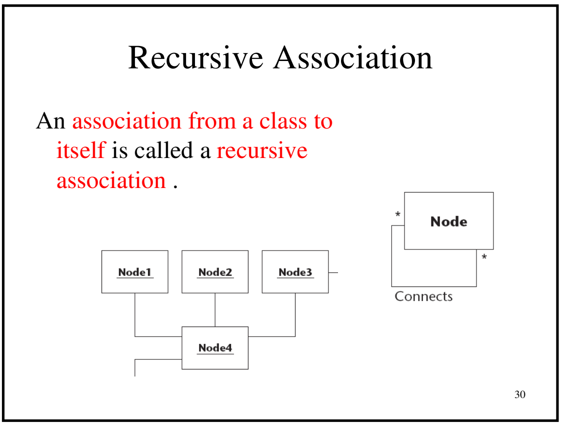
28



Normal Association

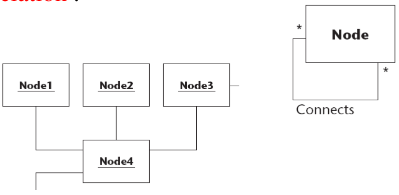


29



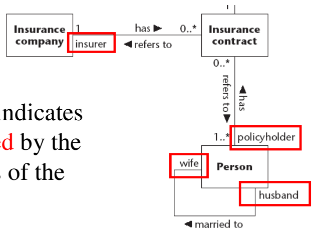
Recursive Association

An association from a class to itself is called a recursive association .



30

Roles in an Association

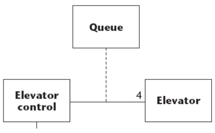


The role name indicates the role played by the class in terms of the association.

31

Association Class

- A class can be attached to an association, in which case it is called an *association class*.



32

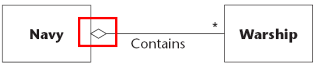
Aggregation

- Aggregation is a special case of association. The aggregate indicates that the relationship between the classes is some sort of "**whole-part**."
- Often describes different levels of abstraction (car consists of wheels, engine, and so on).

33

Shared Aggregation

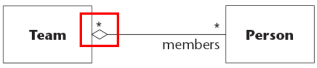
- A shared aggregation is one in which the parts may be parts in any wholes. That an aggregation is shared is shown by its multiplicity.



34

Shared Aggregation (Cont.)

- The aggregation is shared if the multiplicity on the whole side is other than one (1).
- Shared aggregation is a special case of a normal aggregation.



35

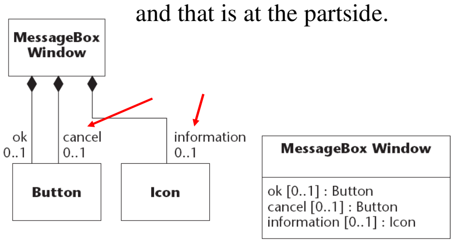
Composition Aggregation

- A composition aggregation **owns** its parts.
- The composition aggregation is one with strong ownership. **The parts "live" inside the whole**; they will be destroyed together with its whole.

36

Composition Aggregation (Cont.)

An aggregate has only one role name, and that is at the partside.



37

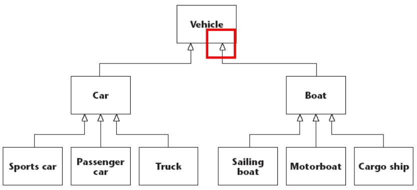
Generalization

UML defines generalization as follows:

- A taxonomic relationship (classified) between a more general classifier and a more specific classifier.
- Each instance of the specific classifier is also an instance of the general classifier.
- Thus, the specific classifier indirectly has features of the more general classifier.

38

Generalization (Cont.)



39

Generalization (Cont.)

- Generalization shows a close relationship between a general and a specific class. The specific class, called the subclass, inherits everything from the general class, called the superclass.
- The attributes, operations, and all associations of the superclass become a part of the subclass.

40

Visibility between Super and Sub

- Attributes and operations with public visibility in the superclass will be public in the subclass as well.
- Members (attributes and operations) that have private visibility will also be inherited, but are not accessible within the subclass.
- To protect attributes or operations from access from outside the superclass and the subclass, you can assign these with protected visibility.

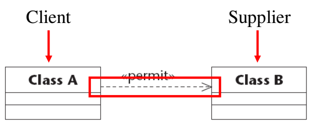
41

Dependencies and Abstractions

- In a dependency relationship, one element represents the **client** that requires a **supplier** element.
- This supplier/client relationship can take on a number of different forms, but in general a dependency indicates that a client is **not complete** without the supplier.

42

Dependencies



43

Common types of dependencies

- uses
- permit
- refine
- trace
- derive

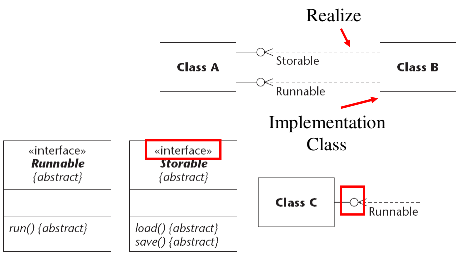
44

Interfaces

- A package, component, or class that has an interface connected to it implements or uses the specified interface by supporting or relying on the behavior defined in the interface.
- An interface cannot include real objects; rather, it contains only abstract operations.
- An interface has a number of signatures that together specify a behavior that any element can support by implementing the interface.

45

Interfaces (Cont.)



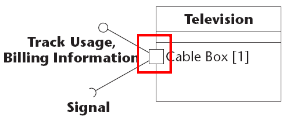
46

Ports

- Ports help model components by showing environmental requirements.
  - The port allows the modeler to insulate (isolate) the inner workings of a class from environmental variables.
  - Such insulation keeps the developer focused on the responsibility of the class without concern for the deployment environment.
  - So long as the deployment environment meets the port's specifications, the component will work.

47

Ports (Cont.)



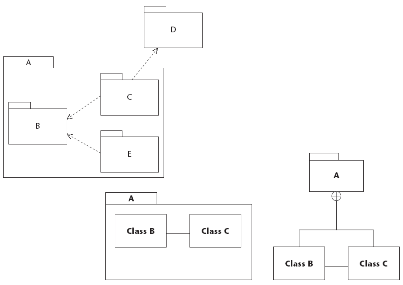
48

Packages

- A package provides a **grouping** mechanism for organizing UML elements.
- A package is used to group elements and to provide a **namespace** for the grouped elements.
- All model elements that are owned or referenced by a package are called the **package contents**.

49

Packages (Cont.)



50

Summary

- UML provides the syntax and semantics to create a model. The modeling language, however, cannot tell you whether you have done a good job. You must strive to make all models easy to communicate, verify, validate, and maintain.

51

Dynamic Modeling

- State machines
- Activity diagrams
- Interaction diagrams

52

Sequence diagrams

- Illustrate how objects interact with each other.
  - They focus on **message** sequences, that is, how messages are sent and received between a number of objects.
- Sequence diagrams have two axes:
  - The vertical axis shows time
  - The horizontal axis shows a set of object

53

Generic and Instance Form

Sequence diagrams can be used in two forms:

- The instance form describes a specific scenario in detail
- The generic form describes all possible alternatives in a scenario

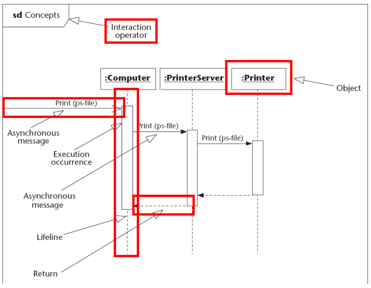
54

Message, Execution occurrence, Lifeline

- A message is a communication between objects that conveys information with the expectation that action will be taken.
  - The receipt of a message is normally considered an event.
- An execution occurrence shows the focus of control.
- The lifeline represents the existence of an object at a particular time.

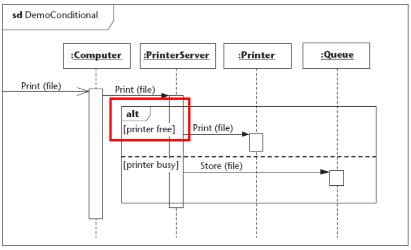
55

The concepts



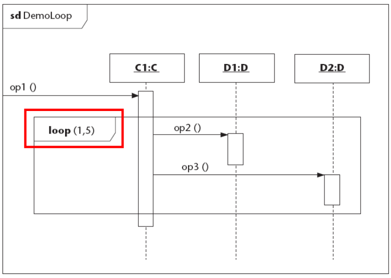
56

Fragment (alt)



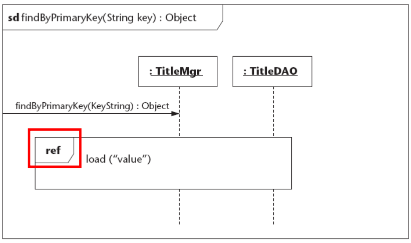
57

Fragment (loop)



58

Fragment (ref)


sd findByPrimaryKey(String key) : Object
: TitleMgr
: TitleDAO
findByPrimaryKey(KeyString) : Object
ref
load ("value")

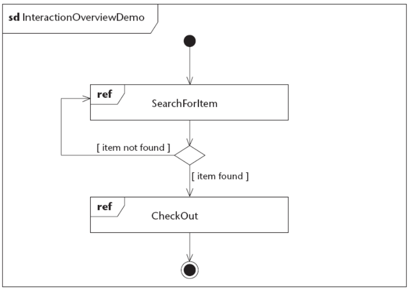
59

Interaction Overviews

- The interaction overview diagram is a variant of an activity diagram.
- Various flow of control nodes from activity diagrams can be combined with sequence fragments to create an interaction overview diagram.

60

Interaction Overviews (Cont.)



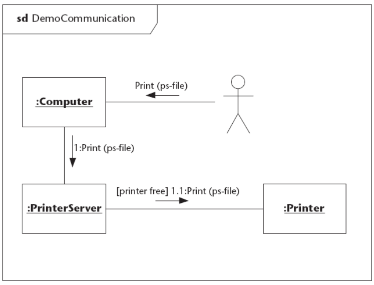
61

Communication Diagrams

- Communication diagrams focus both on the interactions and the links among a set of collaborating objects.
- Communication diagrams show objects and how messages are sent between the linked objects and thereby imply their relations.
- Cannot show structuring mechanisms such as conditional or reference fragments.

62

Communication Diagrams (Cont.)



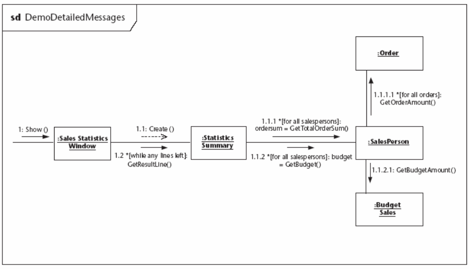
63

Message Labels

- predecessor guard-condition sequence-expression return-value := signature
  - predecessor: sequence-number ‘,’ ... ‘/’
  - guard-condition: ‘[‘ condition-clause ‘]’
  - sequence-expression: [integer | name][recurrence] ‘:’

64

Message Labels (Cont.)


sd DemoDetailedMessages
:Order
1.1.1.1 *[for all orders]: GetOrderAmount()
1: Show ()
:Sales Statistics Window
1.1: Create ()
:Statistics Summary
1.1.1 *[for all salespersons]: ordersum = GetTotalOrderSum()
:SalesPerson
1.2 *[while any lines left]: GetResultLine()
1.1.2 *[for all salespersons]: budget = GetBudget()
1.1.2.1: GetBudgetAmount()
:Budget Sales

5

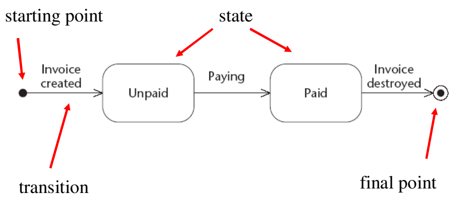
State Machines

UML defines two kinds of state machines:

- Behavioral state machines
  - Behavioral state machines capture the life cycles of objects, subsystems, and systems. They tell the states an object can have and how events affect those states over time.
- Protocol state machines

66

State Machines (Cont.)

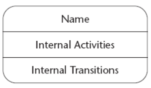


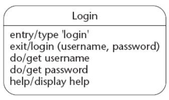
67

State compartments

- Name
- Activity
  Standard event
  – entry
  – exit
  – do

  event-name argument-list '/' action-expression
- Transition





entry/type 'login'
exit/login (username, password)
do/get username
do/get password
help/display help

68

Concepts

- Event-Signature
  - draw (f : Figure, c : Color)
- Guard-Condition
  - [number of invoices > n]
  - withdrawal (amount) [balance >= amount]
- Action-Expression
  - increase () / n := n + 1 / m := m + 1
- Send-Clause
  - left_mouse_btn_down(location) / color:=pick_color(location) ^ pen.set(color)

69

Event types

- A condition becoming true.
- Receipt of an explicit signal from another object.
- Receipt of a call on an operation by another object (or by the object itself).
- Passage of a designated period of time.

70

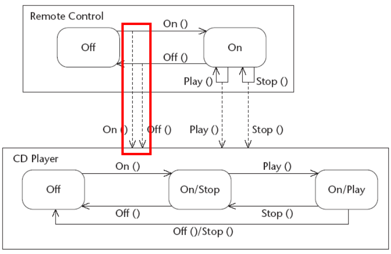
Sending Messages Between State Machines

State machines can send messages to other state machines.

- This process is shown either by actions (send-clause) or
- with dashed arrows between the state machines.

71

Sending Messages Between State Machines (Cont.)



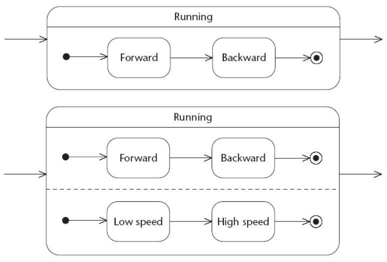
72

Substates

- A state may have nested substates, whereby internally the state has its own substates that can be shown in other state machines.
- UML 2 defines substate machines.
  - orthogonal (or-substates)
  - nonorthogonal (and-substates)

73

Substates (Cont.)



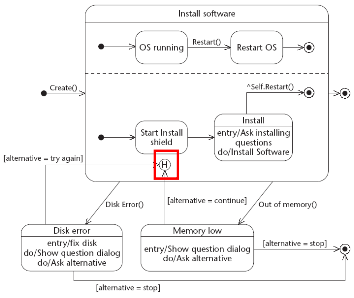
74

History Indicator

- Used to memorize internal states.
- There are 2 types of history indicators:
  - **Shallow history**: the transition is to the most recent state of the immediately enclosing state machine.
  - **Deep history**: Same as shallow history, But the transition is applied recursively to all the enclosing state machines.
- Both have no outgoing transitions.

75

History Indicator (Cont.)



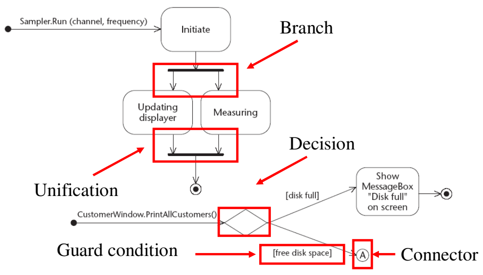
76

Activity Diagrams

- Capture actions and their results.
- Used for different purposes, including:
  - capture the work (actions) that will be performed when an operation is executing.
  - capture the internal work in an object.
  - show how a set of related actions can be performed and how they affect objects around them.
  - show how an instance of a use case can be performed in terms of actions and object state changes.
  - To show how a business works in terms of workers (actors), workflows, organization, and objects

77

Activity Diagrams



78

Activity Diagrams (Cont.)

- Events are attached only to the transition from the start point to the first action.
- Decision
  - Guard-condition
- Branch
- Unification
- Connector

79

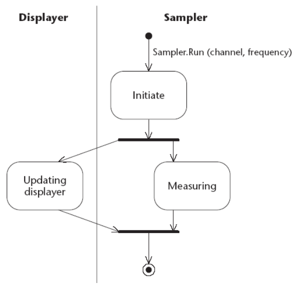
Activity Partitions

Activity partitions group actions, normally with respect to their responsibility.

- To show explicitly where actions are performed (in which object),
- To show in which part of an organization work (an action) is performed.

80

Activity Partitions (Cont.)



81

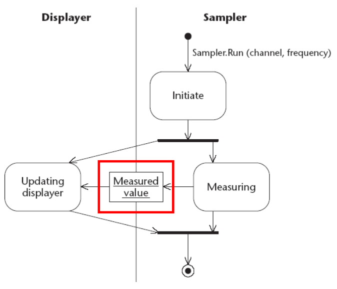
Objects

Objects can be shown on activity diagrams.

- They are either input to or output from the actions,
- or they can simply show that an object is affected by a specific action.

82

Objects (Cont.)



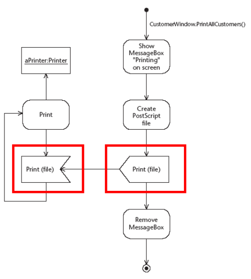
83

Signals

- Signals can be sent or received in activity diagrams
- You use two symbols for signals, one for sending and one for receiving a signal.

84

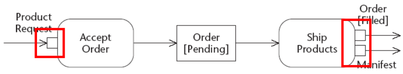
Signals (Cont.)



85

Pins

Shown as a small rectangle attached to an activity, shows values an activity accepts (input pins) and values it produces (output pins).



86