

Requirements Modeling

- Architecture.
- System Interface Prototypes.
- Glossary.
- FURPS.
- Use Case Model.

1

System Boundary

- This is the boundary between the system being developed and the outside world.
- Be careful which system is being bounded. There may be systems within systems.

2

Architecture

- One system may in turn be made up of several subsystems, which interact with each other.
- Each subsystem may be independently developed.
- It is helpful if the subsystems can be arranged in a layered architecture.

3

Identifying Subsystems

- Functions (UI, persistence, security, communication, application, etc.).
- Physical location (mission control, launch control, space shuttle).
- Hardware/OS platform (data acquisition computer, central processor, report server, clients).

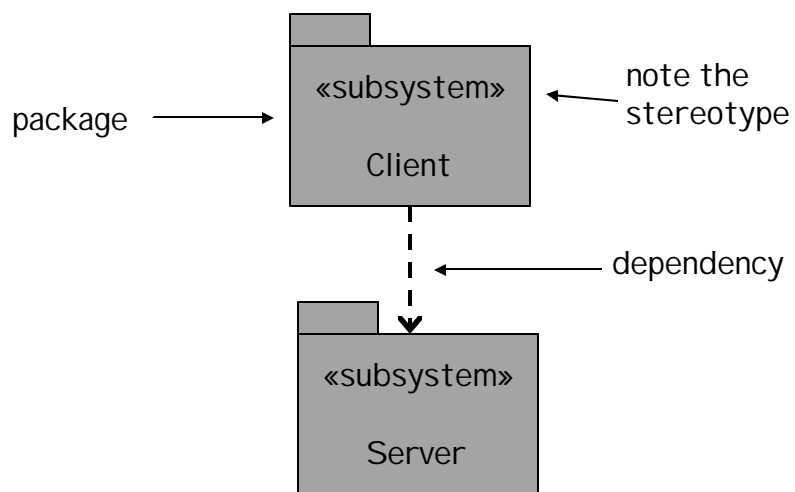
4

Lattice Structure

- In UML, we may represent subsystems as packages, and client/server dependencies as dashed arrows from client to server.
- It is highly desirable that there be no cycles in the dependency graph.

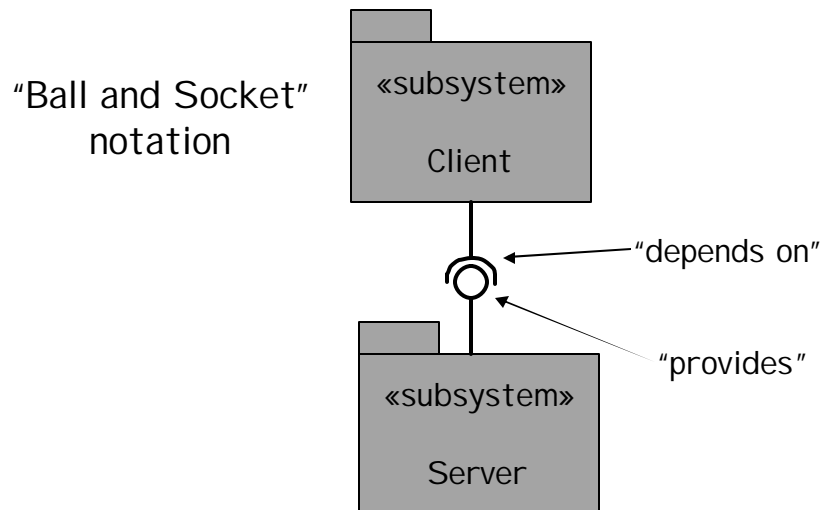
5

UML Subsystem Notation



6

Subsystem Interfaces



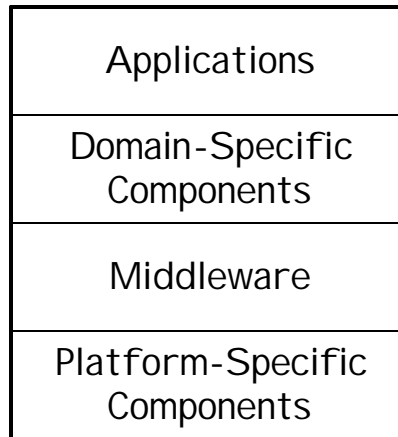
Client-Server Relationship

- Clients know the servers.
- Servers do not know the clients.
- Clients don't know how the servers work.
- Servers don't know how the clients work.

8

Layered Architecture

- Group subsystems into layers based on level of abstraction.
- All dependencies are downward.
- Common layering approach →



9

Subsystem Context

- For each subsystem, we determine its context in the larger system.
- Anything outside the subsystem that interacts with it is called an actor.
 - Other software subsystems.
 - Hardware.
 - People.
 - etc.
- (There is no UML notation for a context diagram.)

10

User Interface Prototype

- We may find it useful to prototype the mechanism for interacting with a system.
- GUI (windows, buttons, menus, list boxes, etc.).
- Menu structures.
- Report layouts.
- Message formats.

11

Uses of the Interface Prototype

- Quick feedback on usability.
- Client involvement in requirements definition.
- Guides analysts and designers.
 - **Caution:** There is a danger that a UI prototype may constrain the analysis and design.

12

Forms of UI Prototype

- Low fidelity – “paper prototype.”
 - May be actually on paper.
 - Or may be on Powerpoint slides.
 - Useful for “storyboarding.”
- High fidelity.
 - Executable mockup.
 - Useful for exploration.

13

Glossary

- We need to keep a list of the definitions of important words.
- Avoid these problems:
 - Name clashes – same name for different concepts.
 - Multiple definitions – different words for the same thing.
 - Missing definitions – no commonly agreed-upon definition for a thing.

14

Hints For Creating a Glossary

- Use the customer's or user's definitions.
- Use a computerized tool.
 - As we go through requirements, analysis and design, the number of things to keep track of will grow substantially.
 - A tool helps with searching for words.

15

Types Of Requirements

- FURPS.
 - Functionality.
 - Usability.
 - Reliability.
 - Performance.
 - Supportability.

16

Supplementary Specification

- URPS.
- Other “ilities” – maintainability, interoperability, portability, availability, manufacturability, etc.
- Design and implementation constraints.
- Documentation requirements.
- Licensing.
- Standards.
- Operational concerns.

17

Functional Requirements

- In the UML, functional requirements are modeled with “Use Cases.”
- There are several levels of abstraction in use case models.
 - Top level use case diagram – use case names, descriptions.
 - Detailed use case diagrams – decomposition.
 - Use case specifications – full description.

18

Actors

- In the top level use case diagram, things that exchange information with the system are called *actors*.
- They are typically represented as stick figures on the diagram (even though they may in fact be hardware or software subsystems, as well as people).

19

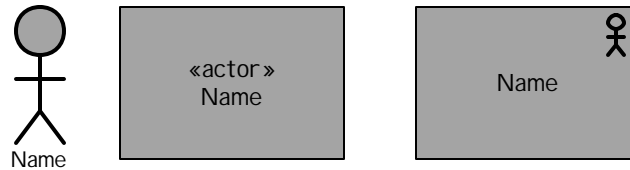
Actors (continued)

- One entity may play the role of different actors at different times. Several people may play the role of the same actor in interacting with the system.
- Actors that initiate actions are sometimes called *primary actors*. Actors that simply respond to messages from the system are called *secondary actors*.

20

Actors Are Classifiers

- There may be any number of instances of a classifier.
- In UML, classifiers are represented as rectangles, or as special icons.
- These are all forms of *actor* classifier:



21

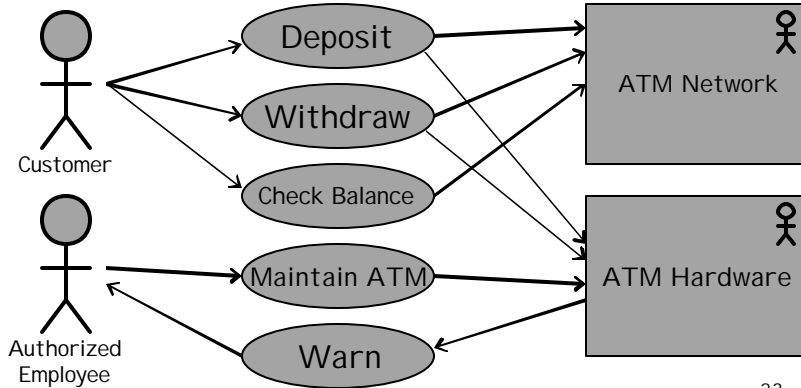
Top Level Use Case Diagram

- Shows the actors that interact with the subsystem.
- Shows the use cases that the system performs.
- Shows associations that indicate:
 - Which primary actors initiate which use cases.
 - Which secondary actors are involved in which use cases.

22

Example Top Level Use Case Diagram

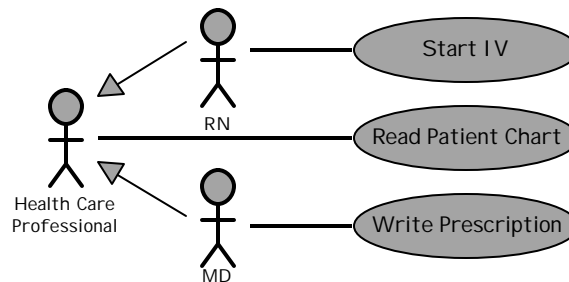
Automated Teller Machine (ATM)



23

Actor Generalization

- Exactly one primary actor per use case.
- You can accomplish this rule if you use actor generalization.



24

Use Case Granularity

- A use case is a complete sequence of interactions between actors and the system that returns an observable result of value (usually to the primary actor).
- Distinguish between “system interactions” and “user goals.”
 - User goal - result of value that an actor receives from performing a use case.
 - One user goal may involve several system interactions.

25

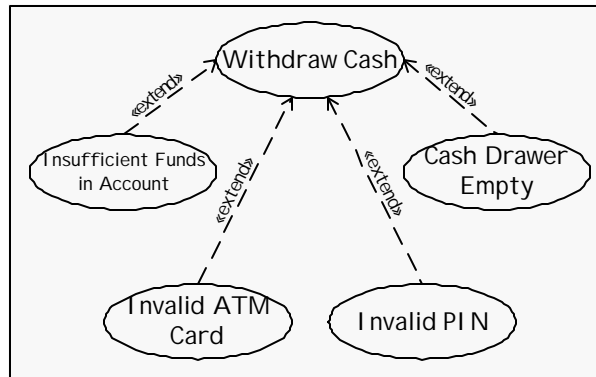
Detailed Use Case Diagrams

- All use cases are represented as ellipses.
- Three kinds of relationships between use cases:
 - Extension.
 - Inclusion.
 - Generalization.

26

Use Case Extension

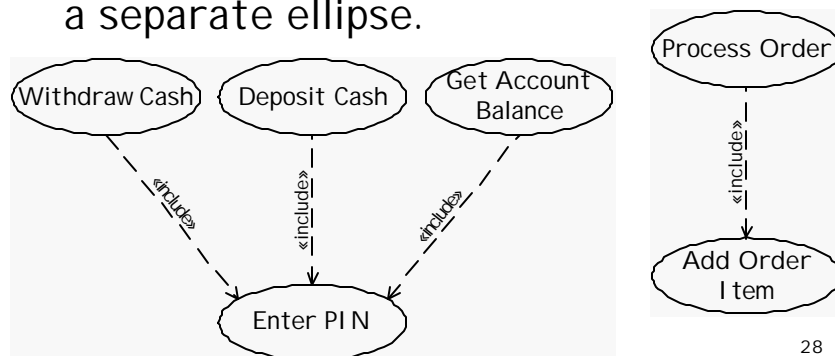
The extension is only performed *some* of the time.



27

Use Case Inclusion

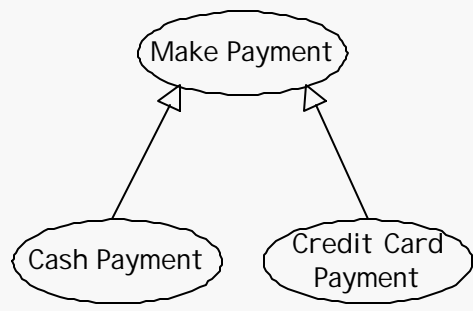
When some sequence of actions is repeated, it may be factored out into a separate ellipse.



28

Use Case Generalization

- Similar to class generalization-specialization (inheritance) relationship.



29

Use Case Descriptions

- To document a use case, we need to specify the following:
 - Use case (short) name.
 - Unique identification (for cataloging).
 - Actors (primary, secondary).
 - Stakeholders.
 - Preconditions.
 - Postconditions.
 - Sequence of events.
 - Special requirements.

30

Preconditions

- Assumed to be true prior to the start of the use case.
- Constraints on the state of the system.
- May be established by another use case.
- Don't need to check for precondition in the use case flow of events.

31

Examples of Preconditions

- The Account Representative has successfully logged on to the system.
- The on-line customer has placed one or more items in the shopping cart.
- The user account has been updated.
- One or more orders have been placed.

32

Postconditions

- Established by the steps in the use case flow of events.
- Guaranteed to be true after completion of the use case.

33

Examples of Postconditions

- The customers account balance is adjusted and the money is dispensed.
- The new account is added to the accounts database.
- The payment is received and the shipment is scheduled.
- The order is fulfilled.

34

Flow of Events

- Sequence of events.
- Events must be observable at the system boundary.
- Possible events:
 - Message from actor to system.
 - Message from system to actor.
 - Action taken by the system (as long as you can somehow detect that the system has taken the action).

35

Narrative Style for Describing Flow Of Events

- Express each event as a complete sentence whose subject is either the system or an actor.
- For example:
 - Customer submits the credit card number.
 - ATM validates that the customer has no outstanding loan payments.

36

Use Case Flows

- Assume precondition.
- Establish postcondition.
- May contain (limited) branching and looping.
 - Branching and looping must result in the same postcondition.
 - Avoid “writing the program” in the use cases.

37

Branching vs «extend»

- If the branching results in a different postcondition, then you can break that functionality out as an «extend» related use case.
- The base use case must contain a step to used as an anchor (or extension point) for the extension.

38

Example of an Anchor

- (in the "Customer checks out" use case of the on-line store):
 - [validate payment] The system successfully validates the payment on the credit card.
- (in the ATM example):
 - [check cash drawer] The system determines that there is sufficient currency in the cash drawer.

39

The «extend» Use Case

- In the «extend» use case, the first line is usually a reference to an anchor step in the base use case.
- For example, in the ATM "Withdraw cash" use case, the first line in the "Cash drawer empty" extension use case, would be:
 - At step [check cash drawer], the system determines that there is insufficient currency in the drawer to process the transaction.

40

Hint: Use Hyperlinks To Refer To Use Case Steps

- Avoid numbering the steps of a use case flow, and then referring to a step by number in an extend use case.
- Use bullets (ordered list).
- In MS Word, use a *bookmark* to indicate an anchor.
 - Word puts brackets around the tag.
 - Hyperlinked references may be made to the tag from other points in the document (or even other documents, if they are in the same master document).

41

Looping vs «include»

- If there are many steps in the loop body, it is a better idea to detail these steps in another use case.
- Use the «include» relationship.
- The entire loop is a single step in the base use case.

42

Example of Looping

- (in the on-line store example):
 - (*in the base use case, one step might be*):
 - "The customer adds one or more items to the shopping cart." *or even,*
 - "The customer performs the 'add item to shopping cart' use case one or more times."
 - The name of the «include» use case would be something like "Add item to shopping cart."

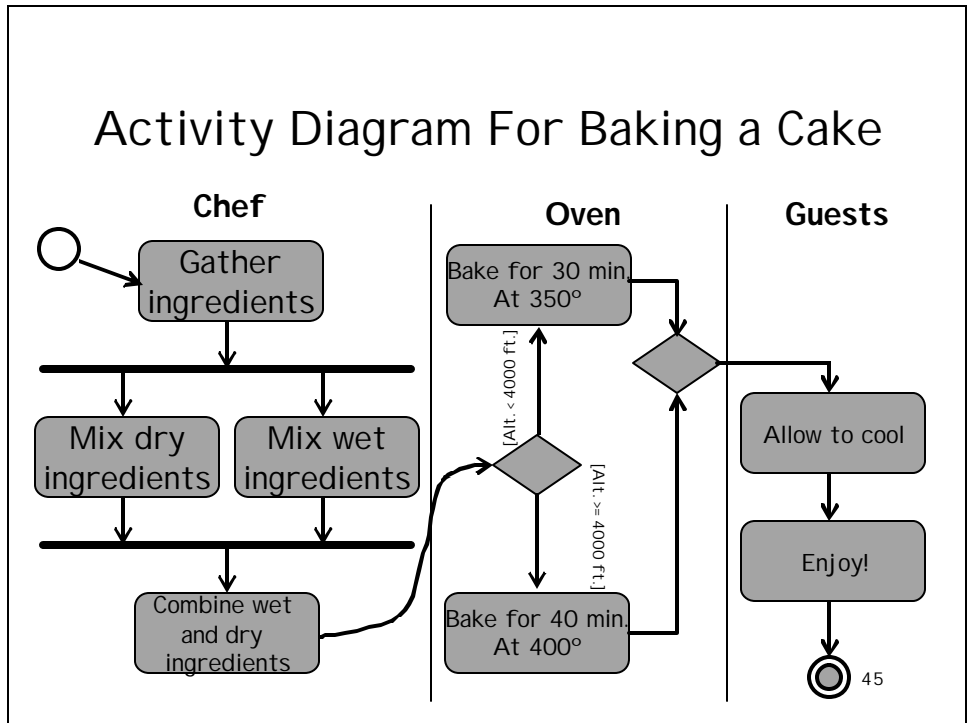
43

An Activity Diagram May Be Used To Show Flow Of Events

- UML Activity Diagram notation shown on next page:
 - Start.
 - Activity.
 - Edge (or transition, or flow).
 - End (final state).
 - Fork.
 - Join.
 - Decision.
 - Merge.
 - Guard.
 - Partition.

44

Activity Diagram For Baking a Cake



Additional Activity Diagram Notation

- Notation not shown on previous page.



- Rake (reference to a nested activity diagram).



- Object node (to show information flow).



- Flow final node (to show termination of a flow, but not the entire activity).

- Looping.

Fork/Join Semantics

- All paths coming from a fork must all be joined in the same join.
 - Unless a path terminates (shown by a "flow final" node).
- Control will not pass beyond a join until all paths either arrive at the join point or terminate.

47

Activity Diagram Structuring Conventions

- Fork/Join structures should be nested.
- Decision/Merge structures should be nested.
- The diagram should be *planar* – you should be able to draw it on a 2 dimensional surface with no crossing flow lines.

48

System State

- For any use case, the system starts in a known state (i.e., the precondition).
- Each step in a use case description assumes a precondition for that step, and leaves the state in a known postcondition for that step.
- By induction, we should be able to determine the state of the system at any point in a use case.
 - After the last step, the system state should imply the postcondition.

49

Semantics of «include»

- An included use case inherits the conditions from the parent use case *at the point of inclusion*.
- The postcondition of the included use case is the new state of the parent use case after the execution of the included use case.

50

Semantics of «extend»

- The state of the system *at the point of the extension* is assumed to be a precondition of the extend use case.
- If there is a stated precondition in the extend use case, it is “anded” to the assumed precondition.

51

General Notes on Structuring

- Use looping and if-then-else language for minor variations in behavior.
- Use activity diagrams for documenting major variations in behavior.
- Use separate use cases (with extend, include and generalization) when you expect to implement the variations in behavior in separate iterations of your iterative process.

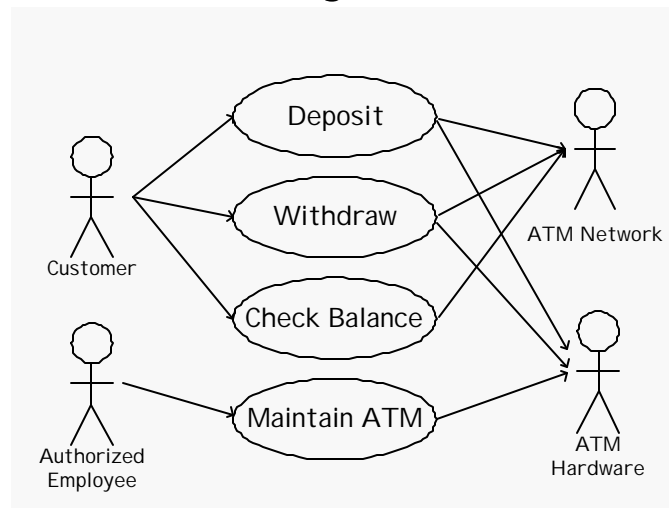
52

Example

- The pages that follow illustrate the documentation of an Automated Teller Machine (ATM).

53

ATM Top Level Use Case Diagram



54

ATM Glossary Entries

Account: A customer's bank account that is linked to an ATM card.

ATM Card: A means of identifying a customer's accounts.

Cash drawer: The mechanism in the ATM that dispenses cash to the customer.

Customer: An account holder at the bank. A customer is identified by swiping a card and entering a PIN.

PIN: Personal Identification Number – a 4 digit number, taken together with the identifying information on the card authenticates a customer.

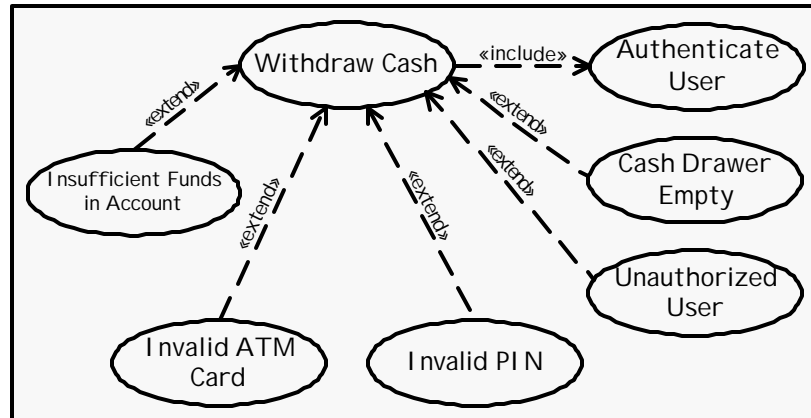
55

Supplementary Specifications

- **Usability:** the average user should be able to learn the interface in five minutes or less, and should make fewer than five mistakes per year.
- **Performance:** The response time for displaying information on the screen shall be less than $\frac{1}{2}$ second.
- **Performance:** The response time for interacting with the ATM network shall be less than 10 seconds.
- **Reliability:** The mean time to failure of the software system shall be at least 60 months.
- **Availability:** The system should operational at least 23 hours per day.

56

Detailed Use Case Diagram for Withdraw Cash



57

ATM Use Case Description (1 of 2)

Name: Withdraw Cash

ID: ATM010

Actor: Customer*, ATM Network, ATM Hardware

Preconditions: ATM is operational, and communication to ATM network is up and running.

Postconditions: Customer successfully withdraws cash from account, and the cash is dispensed by the ATM.

Special conditions: The amount must be a multiple of the smallest currency denomination in the ATM.

58

ATM Use Case Description (2 of 2)

Flow of Events:

- System authenticates user.
- Customer selects "Withdraw" and enters the desired account and the amount.
- [authenticate user] System authenticates the customer.
- [debit account] System successfully debits the account.
- [hardware operational] System opens the cash drawer and dispenses the proper currency.

59

«extend» Use Case Description (1 of 2)

- Name: Insufficient funds in account
- ID: ATM011
- Preconditions: Customer tries to withdraw money from an account with insufficient funds.
- Postcondition: Account balance is unchanged.

60

«extend» Use Case Description (2 of 2)

- Flow of events:
 - At step [debit account] in the basic course, the system determines that there is insufficient balance in the account.
 - The balance is unchanged
 - System indicates the failure to the customer, and the use case ends.

61

Exercise

- Draw an activity diagram for the “withdraw cash” use case.

62