Use case Modeling

Definitions

- **Actor**

  Role that a user plays in the system.

  An actor is a person, organization, or external system that plays a role in one or more interactions with our system.

- **3 kinds of actors**

  1. Primary actor

  Has user goals fulfilled through using services of the system under discussion. Used to find user goals .

  2. Supporting actor

  Provides a service to the suD . Used to clarify external interfaces and protocols.

3. Offstage actor

   Has an interest in the behavior of the use case .

- **Scenario**

  Is a specific sequence of actions and interactions between actors and the system.

  Also called as use case instance.

  Example

  A customer arrives at a checkout with items to purchase. The cashier uses the POS system to record each purchased item. The system presents a running total and line item details. The customer enters payment information , which the system validates and record. The system updates inventory . The customer receives a receipt from the system and then leaves with the items.

- **Use case**

  Is a collection of related success and failure scenarios that describe an actor using a system to support a goal.

- **Example**

  Use case Name : Handle Returns

  Main success scenario

  A customer arrives at a checkout with items to purchase. The cashier uses the POS system to record each purchased item. The system presents a running total and line item details. The customer enters payment information , which the system validates and record. The system updates inventory . The customer receives a receipt from the system and then leaves with the items.

Alternate scenarios:

If the customer paid by credit , and the reimbursement transaction to their credit account is rejected , inform the customer and pay them with cash.

If the system failure to communicate with the external accounting system.

If the item identifier is not found in the system , notify the cashier and suggest manual entry of the identifier code.

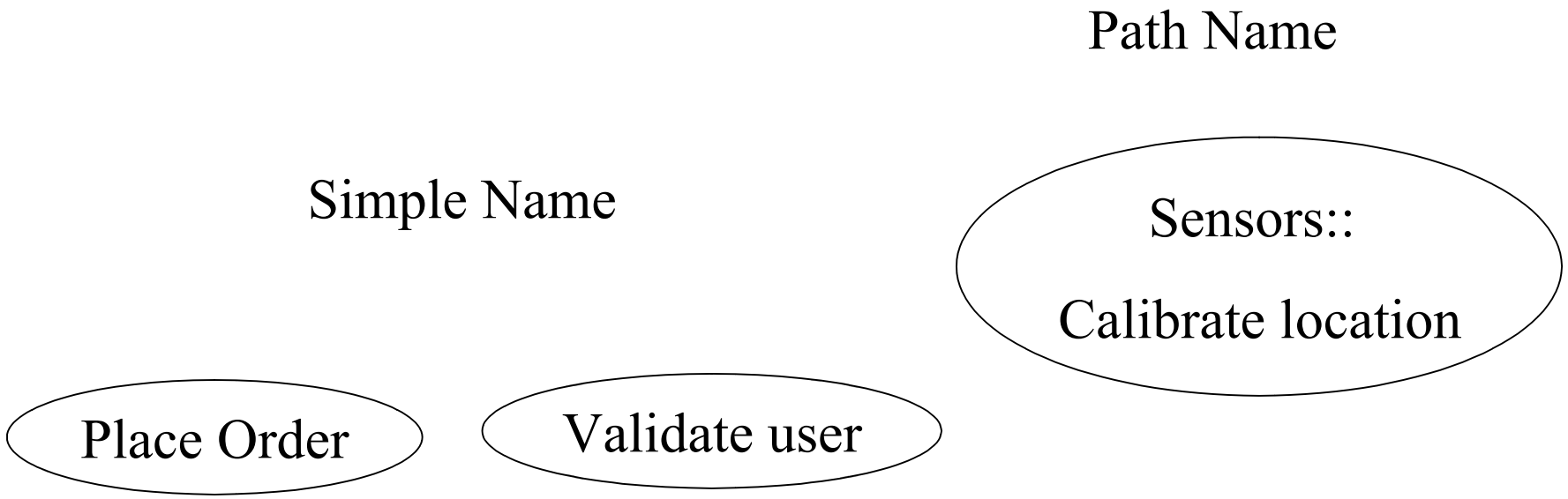
Use Case Diagram

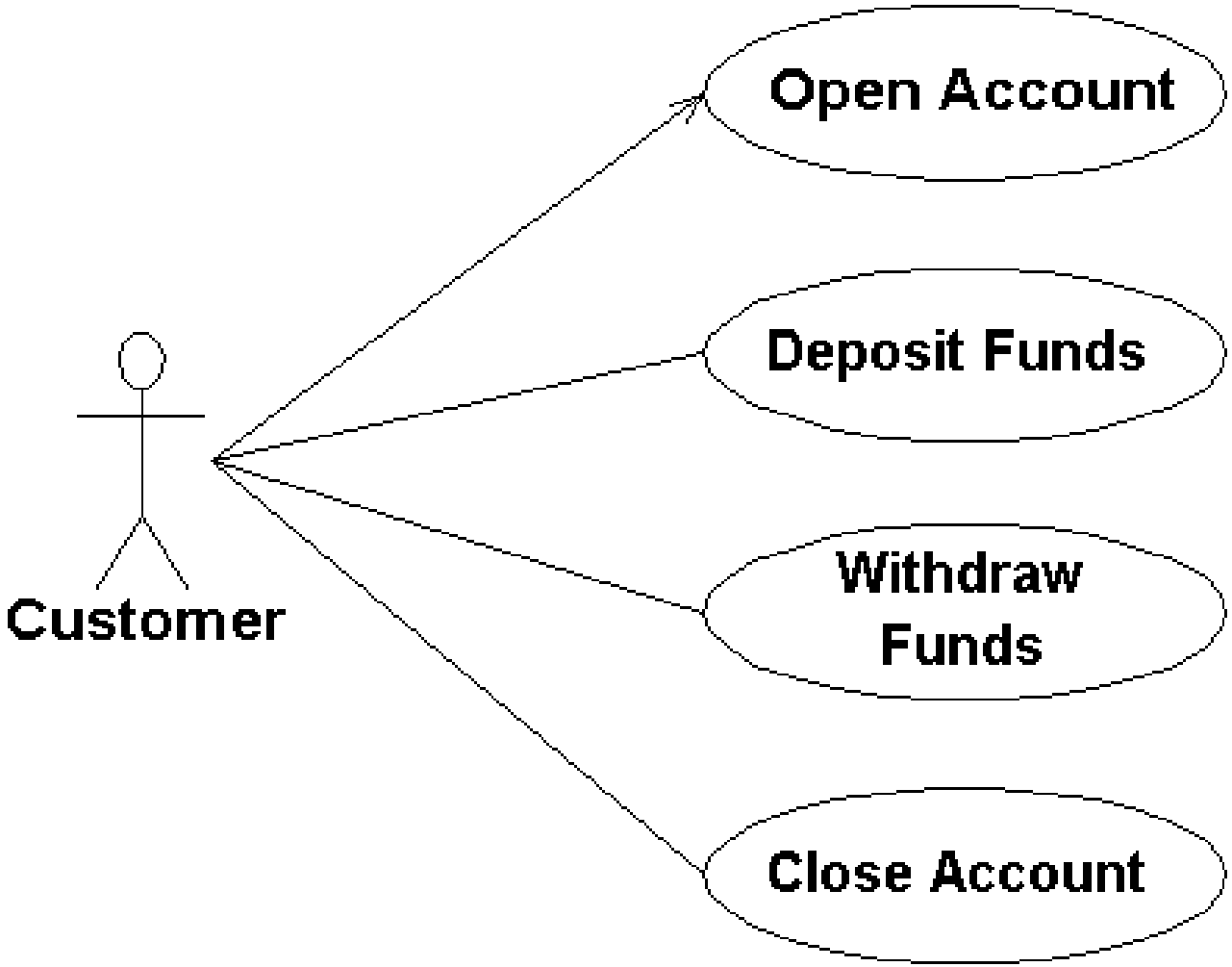
- A use case diagram is a diagram that shows a set of use cases and actors and their relationships.
- Use case concept was introduced by Ivar Jacobson in the object oriented software engineering (OOSE) method.

Use case naming rules

- Every use case must have a name that distinguishes it from other use case.
- A name is a textual string (simple name)
- Use Case Names Begin With a Strong Verb
- Name Use Cases Using Domain Terminology.
- A path name is the use case name prefixed by the name of the package is which the use case lives.
- May consist of any number of letters, numbers and punctuation marks (except colon) and may continue over several lines.

Use case names




Open Account
Deposit Funds
Withdraw Funds
Close Account
Customer

Guidelines

- Actors may be connected to use cases only by association.
- An association between as actor and a use case indicates that the use case and the actor communicates with one another, each one possibly sending and receiving messages.
- Actors don’t interact with one another.

Use case and Flow of Events

- A use case describes what a system (or a subsystem, class or interface) does but it does not specify how it does it.
- We can specify the behavior of a use case by describing a flow of events in text clearly enough for an outsider to understand it easily.
- When we writs this flow of events we should include how and when the use case starts and ends, when the use case interacts with the actors and what objects are exchanged and the basic flow and alternative flows of the behavior.
- Ex. ATM System
- Main Flow of Events: The use case starts when the system prompts the customer for a PIN number.
- Exceptional flow of Events: The customer can cancel a transaction.
- We can specify a use case's flow of events in a number of ways, including informal structural text, formal structural text (with pre and post conditions) & pseudo code.

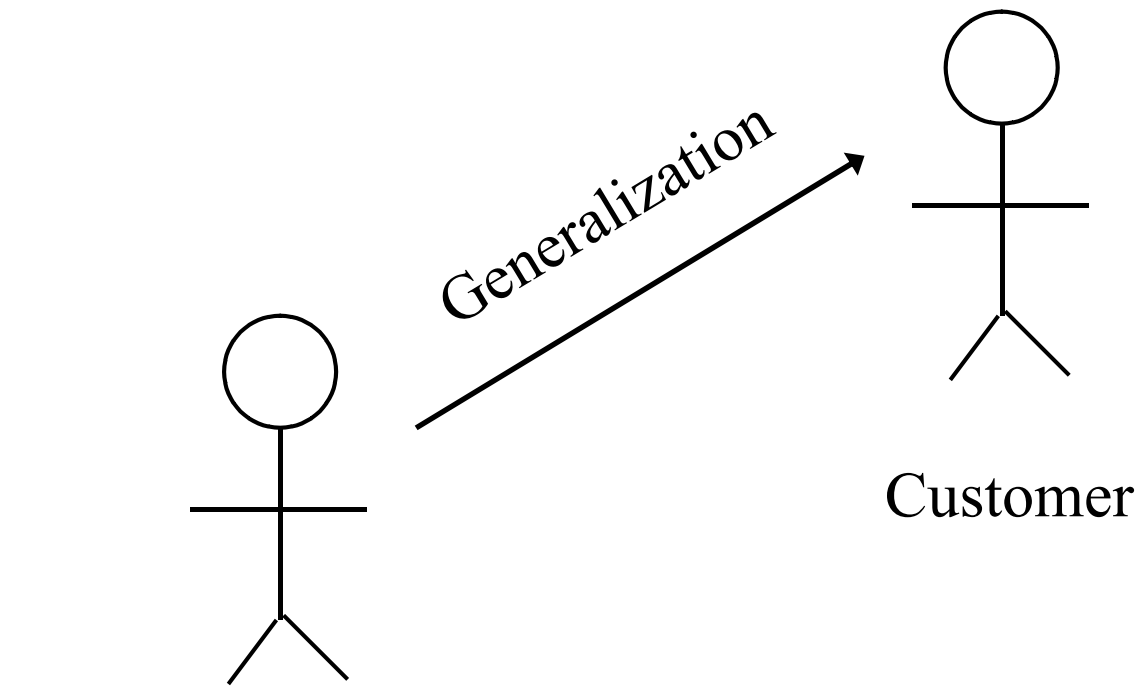
Organizing use case

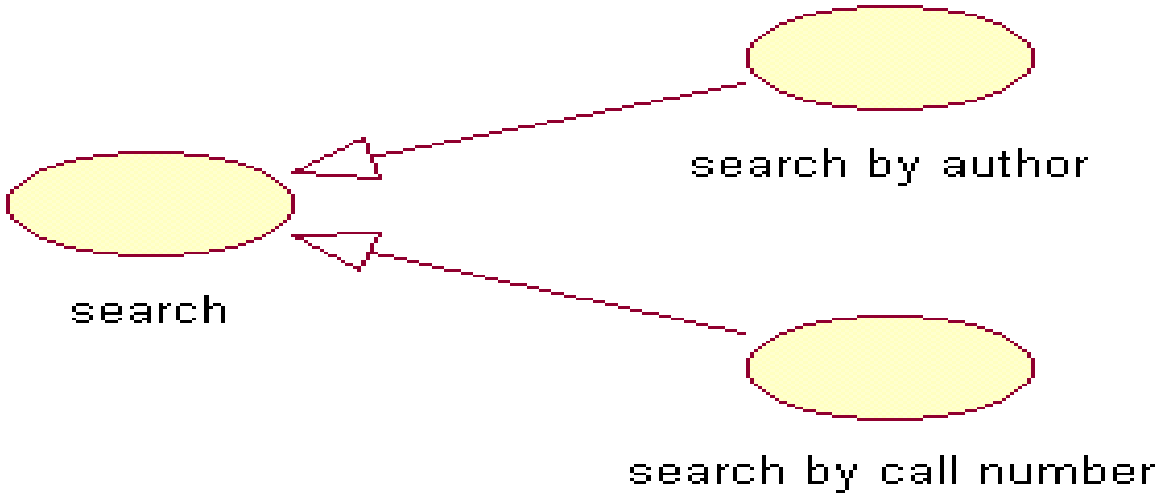
- We can organize use cases by grouping them in packages in the same manner in which we can organize classes.
- We can also organize use cases by specifying generalization include and extend relationships among them.

Generalization

- The child use case inherits the behavior and meaning of the parent use case.
- The child may add to or override the behavior of its parent and the child may be substituted any place the parent appears.
- Ex. Banking system
  - we have the use case validate user
  - Responsible for verifying the identity of the user
  - Two specialized children of this use case.
    - Check password – checking textual password
    - Retinal scan – unique retinal pattern of the user.
  - Both of which behave just like validate user and may be applied anywhere validate user appears.
  - Generalization among use case is rendered as a solid directed line with a large arrowhead.

Generalization

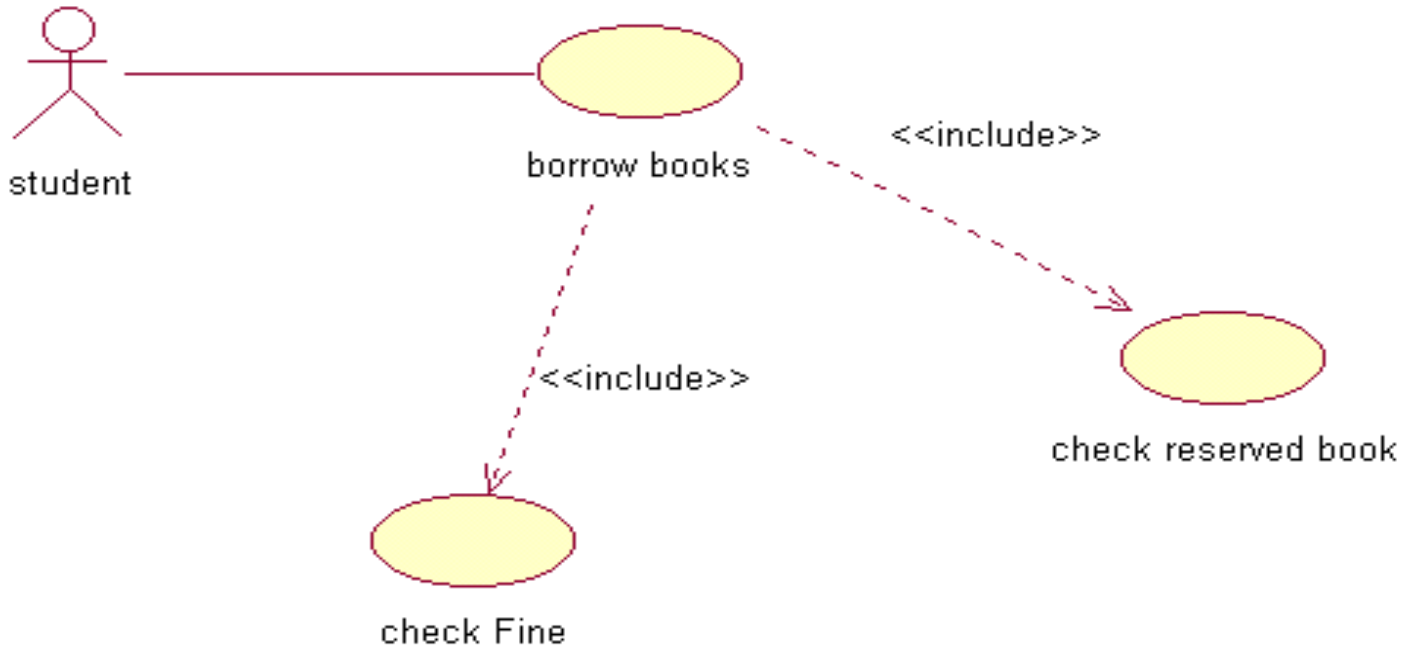



search by author
search
search by call number

Include

- An include relationship between use cases means that the base use case explicitly incorporates the behavior of another use case at a location specified in the base.
- The included use case never stands alone. Use an include relationships to avoid describing the same flow of events several times.
- Simply write include followed by the name of the use case we want to include.

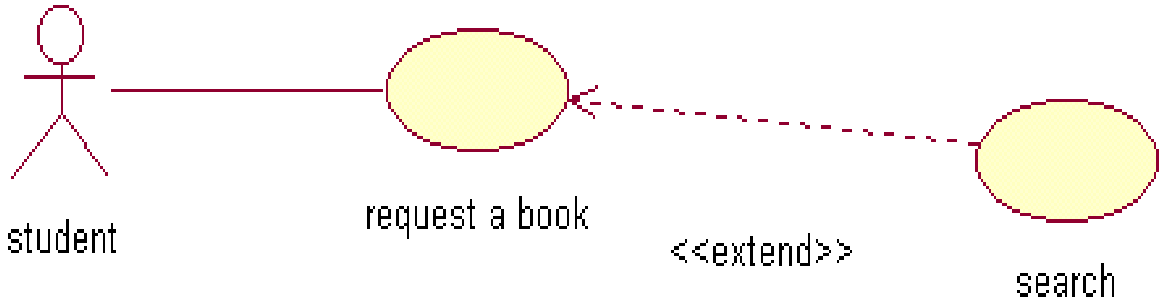
Example


student
borrow books
<<include>>
<<include>>
check reserved book
check Fine

Extend

- An extend relationship between use cases means that the base use case implicitly incorporates the behavior of another use case at a location specified indirectly by the extending use case.
- This base use case may be extended only at certain points, called its extension points.

Example



3 Common use case formats

1. Brief

   One paragraph summary , usually of the main scenario .

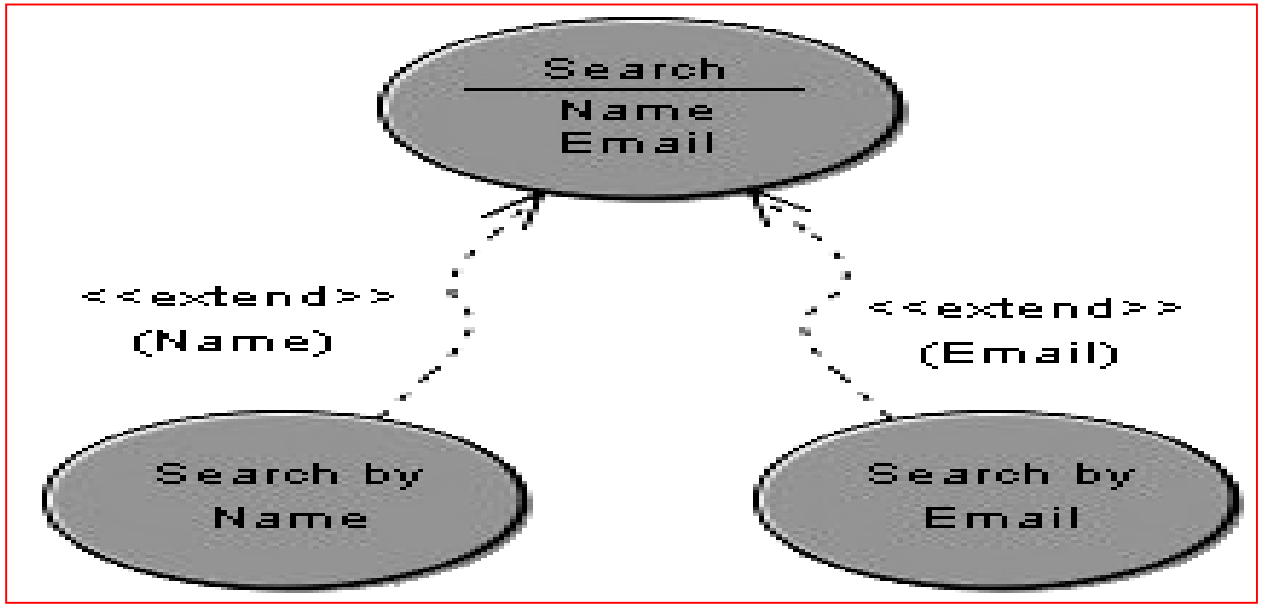
2. Casual

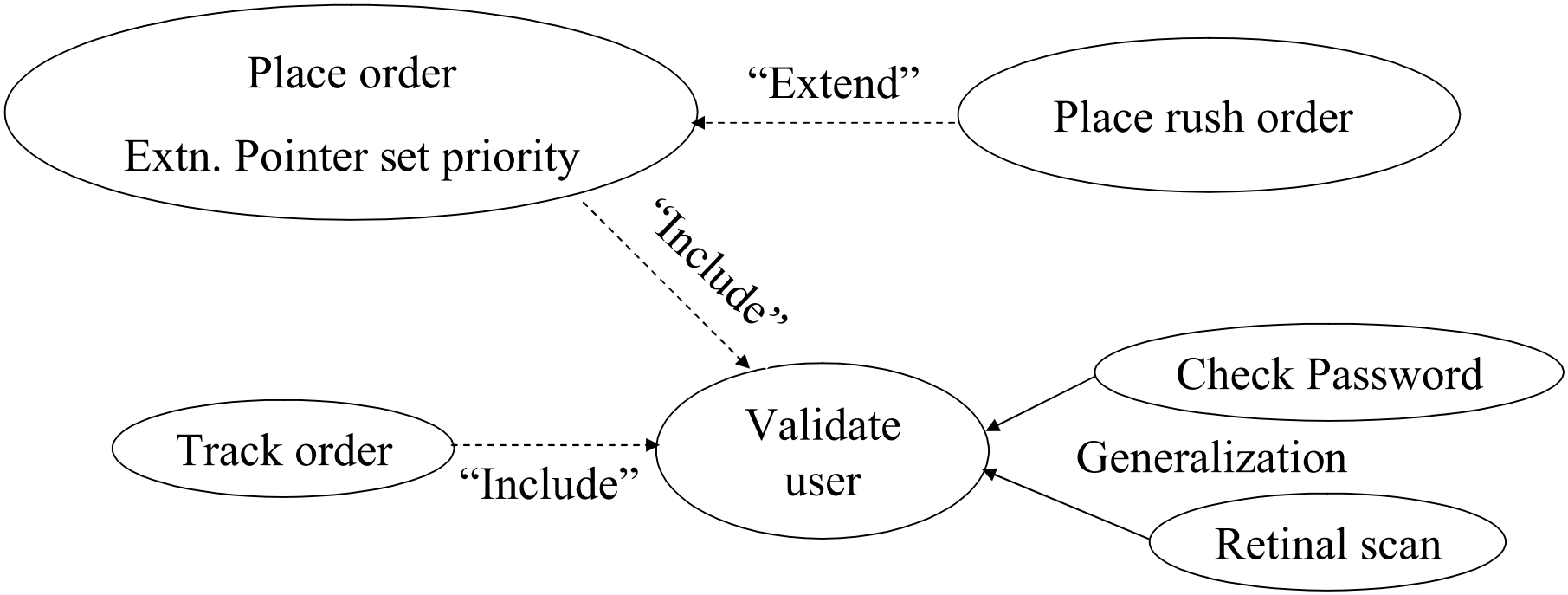
   Informal paragraph format. Multiple paragraphs that cover various scenarios .

3. Fully Dressed

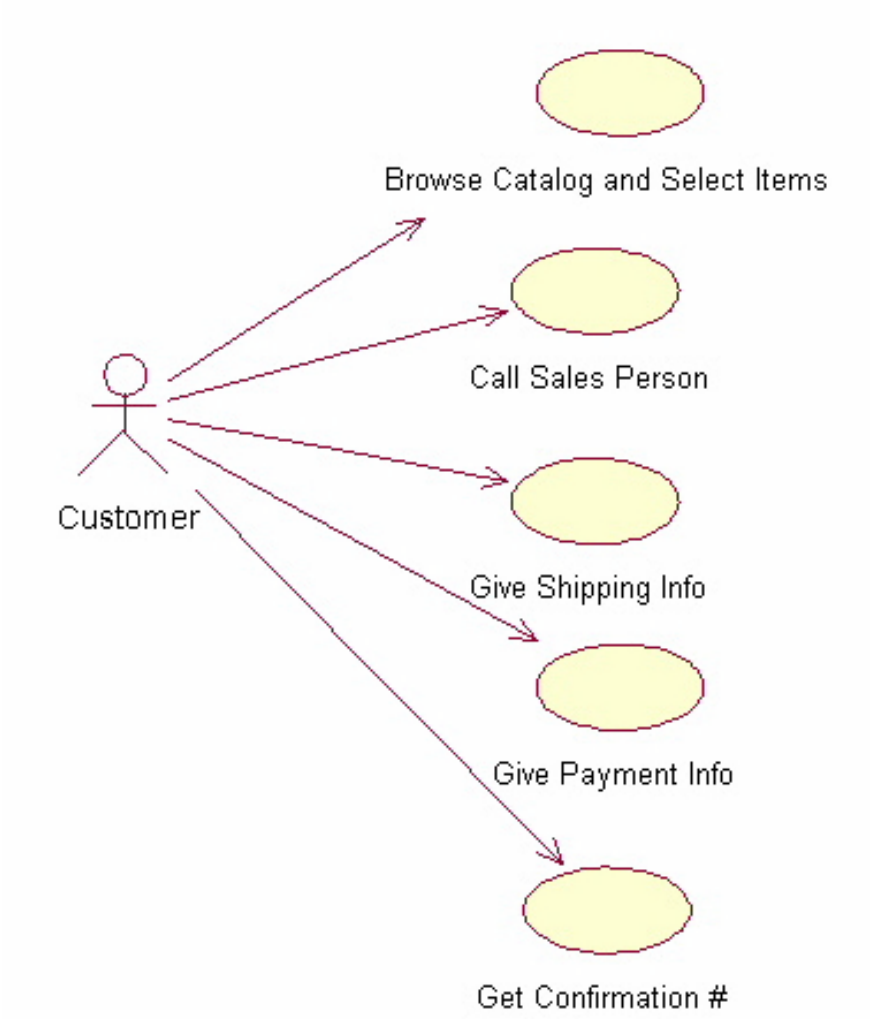
   All steps and variations are written in detail , and there are supporting sections , such as preconditions and success guarantees.

Use Case Section	Comment
Use case Name	Start with a verb
Scope	The system under design
Level	User goal or sub function
Primary actor	Calls on the system to deliver its services
Stakeholders and interests	Who cares about the UC and what do they want ?
Preconditions	What must be true on start .
Success Guarantee	What must be true on successful completion
Main success scenario	Unconditional happy path scenario
Extensions	Alternate scenarios of success or failure
Special requirements	Related non functional requirements
Technology and data variations list	Varying I/O methods and data formats
Frequency of occurrence	Influences investigation, testing and timing of implementation
Miscellaneous	Such as open issues


Search
Name
Email
<<extend>>
(Name)
<<extend>>
(Email)
Search by
Name
Search by
Email


Place order
Extn. Pointer set priority
"Extend"
Place rush order
"Include"
Check Password
Validate user
Generalization
Track order
"Include"
Retinal scan

Simple use case diagram



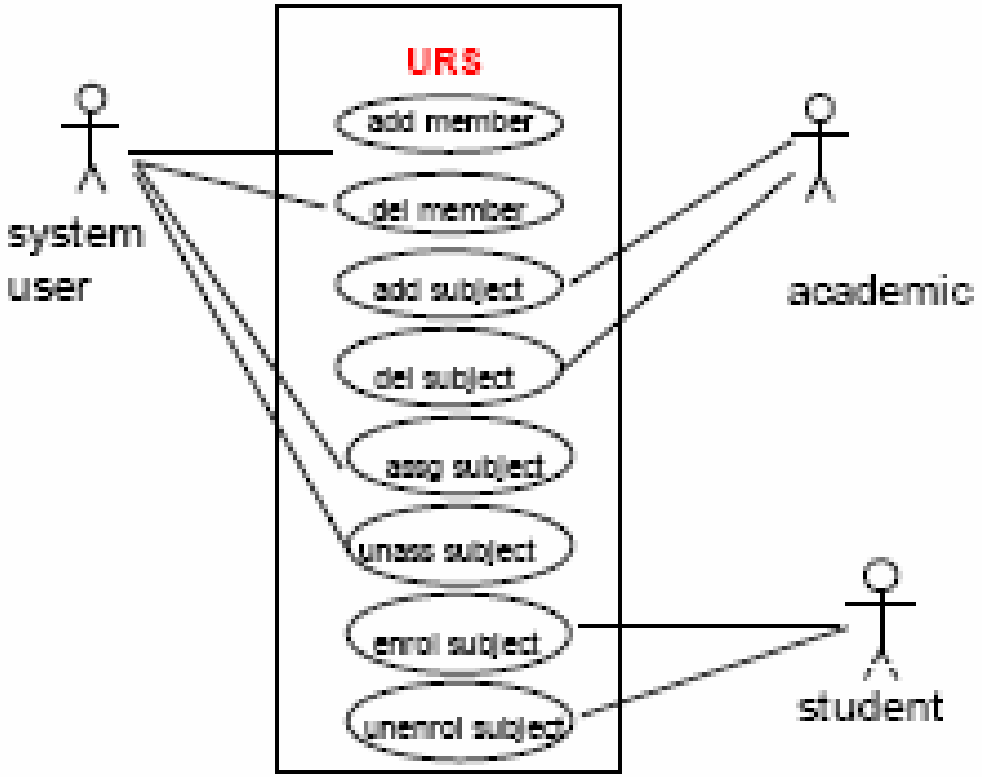
University Record System (URS)

- A University record system should keep information about its students and academic staff.
- Records for all university members are to include their id number, surname, given name, email, address, date of birth, and telephone number.
- Students and academic staff each have their own unique ID number: studN (students), acadN (academic employee), where N is an integer (N> 0).
- Students will also have a list of subjects they are enrolled in. A student cannot be enrolled in any more than 10 subjects. Academic employees will have a salary, and a list of subjects they teach. An academic can teach no more than 3 subjects.

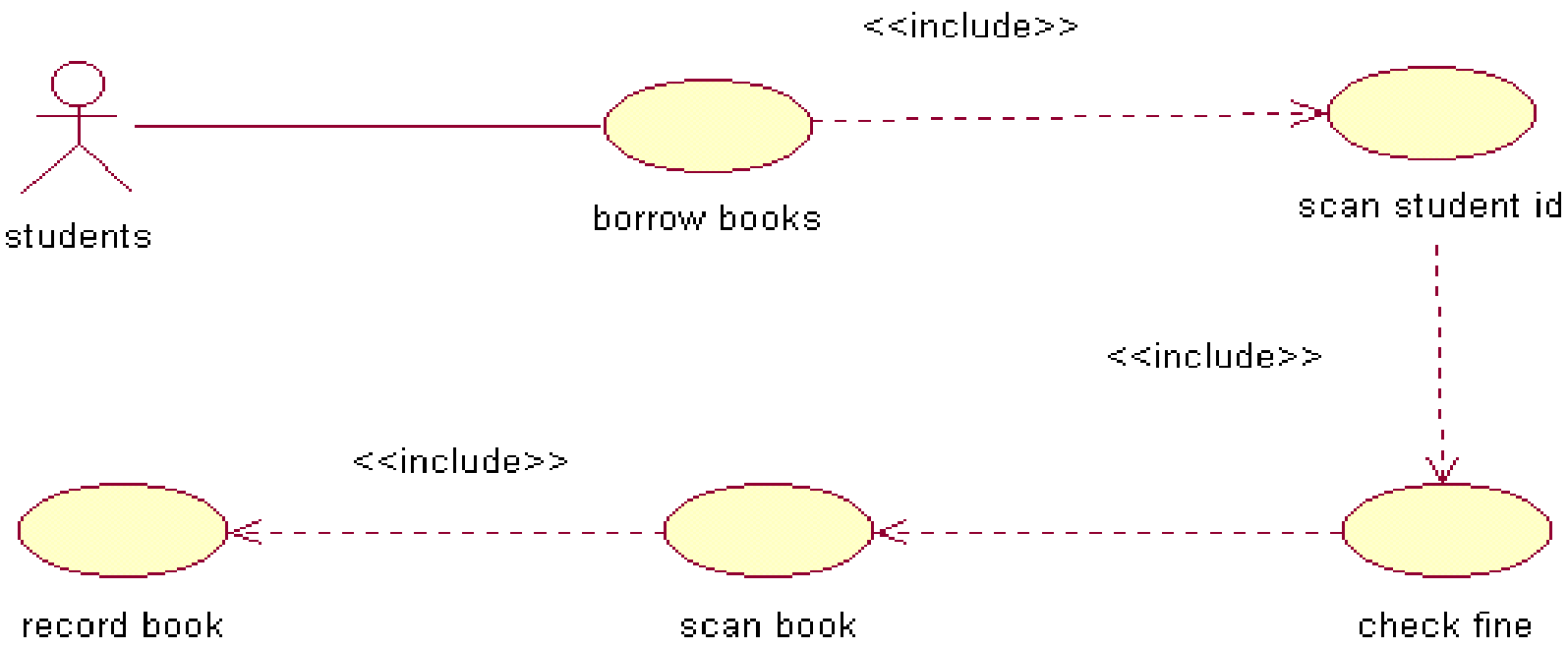
Some Actions Supported by URS

- The system should be able to handle the following commands.
- Add and remove university members (students, and academic staff)
- Add and Delete subjects
- Assign and Un assign subjects to students
- Assign and Un assign subjects to academic staff.

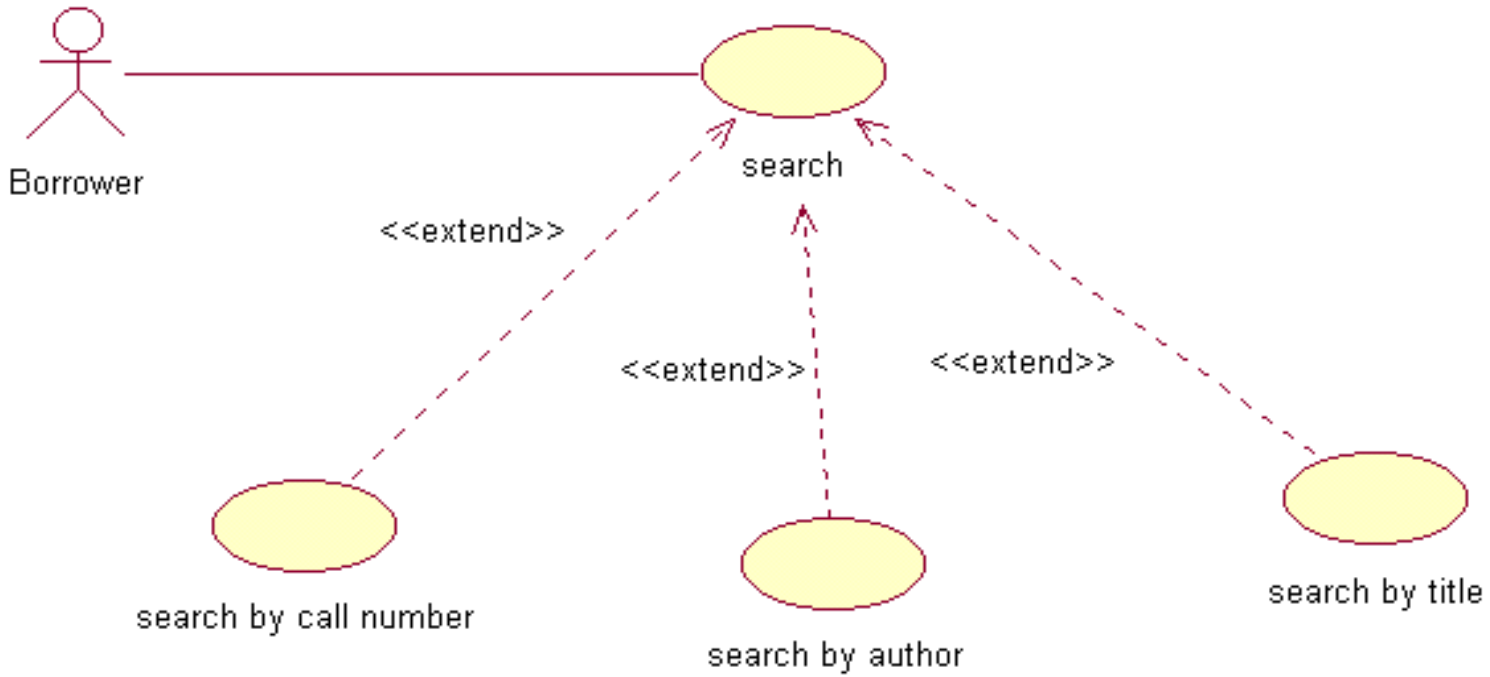
Use Case Diagram - URS System



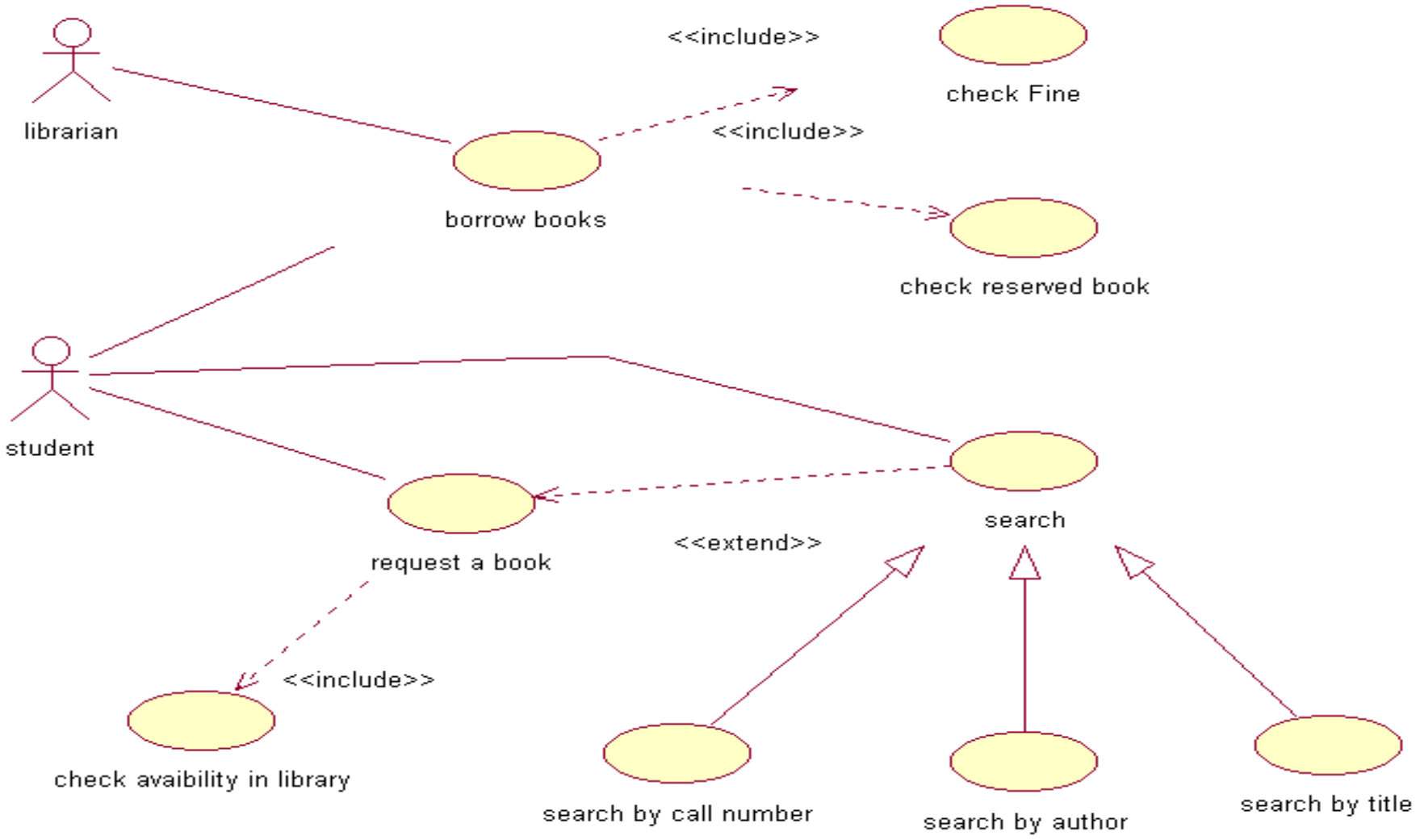
Bad Use case Diagram



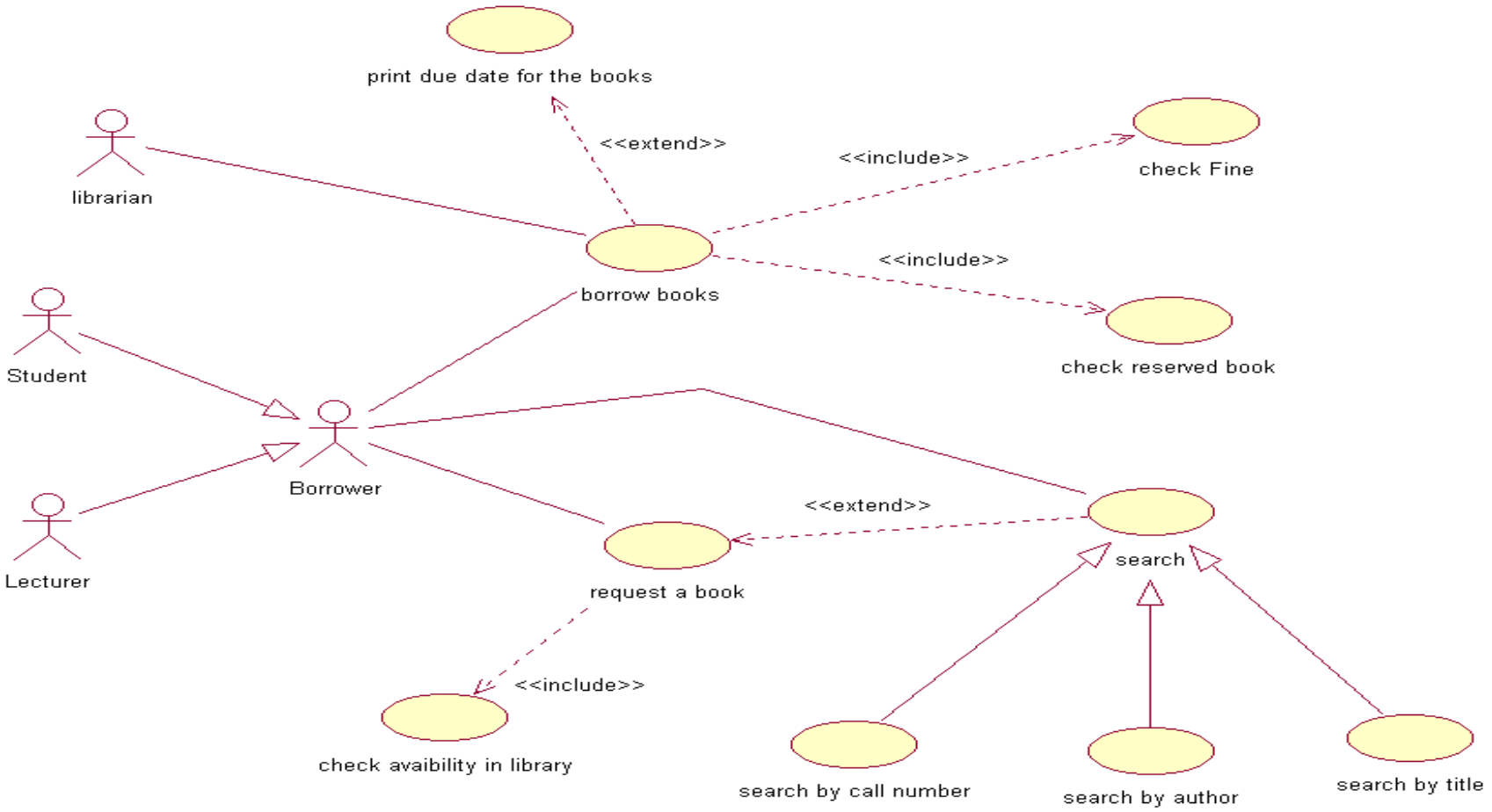
Bad Use case Diagram



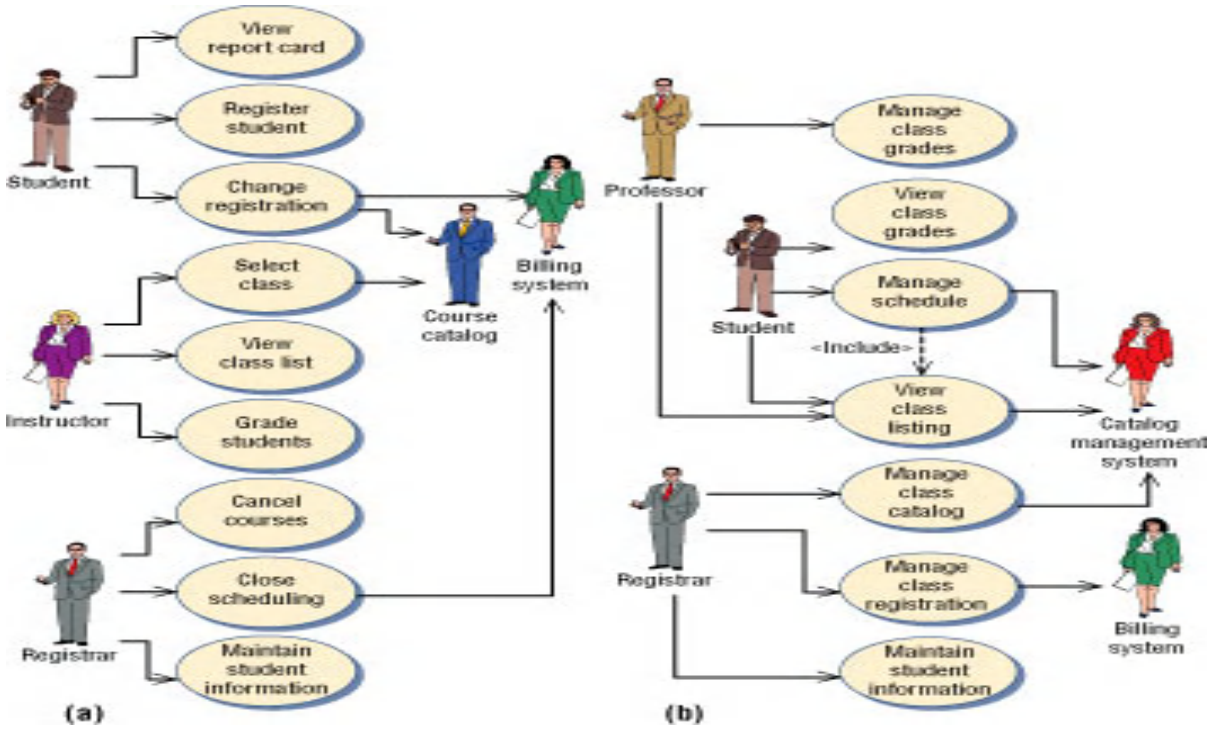
Sample use case diagram



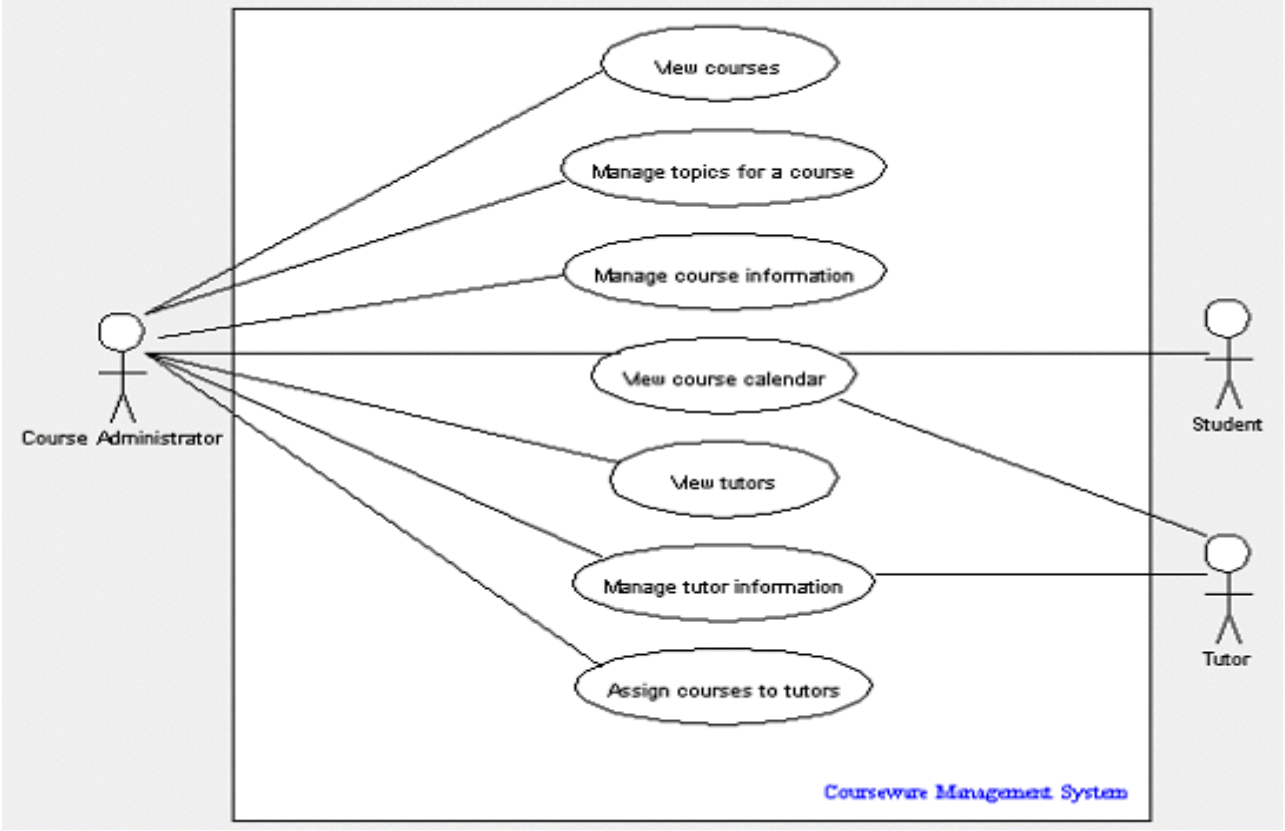
Sample use case diagram

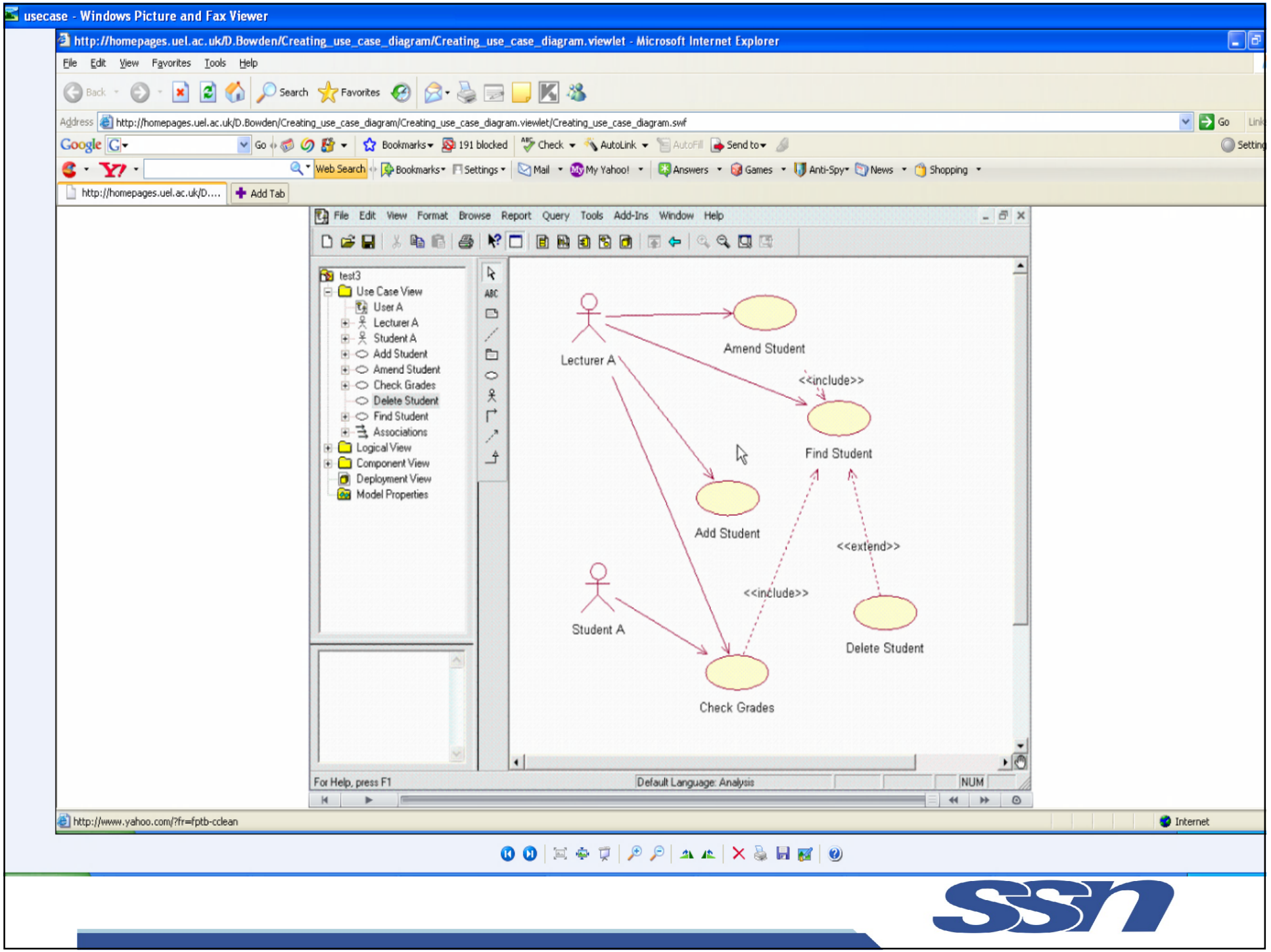


Sample use case diagram

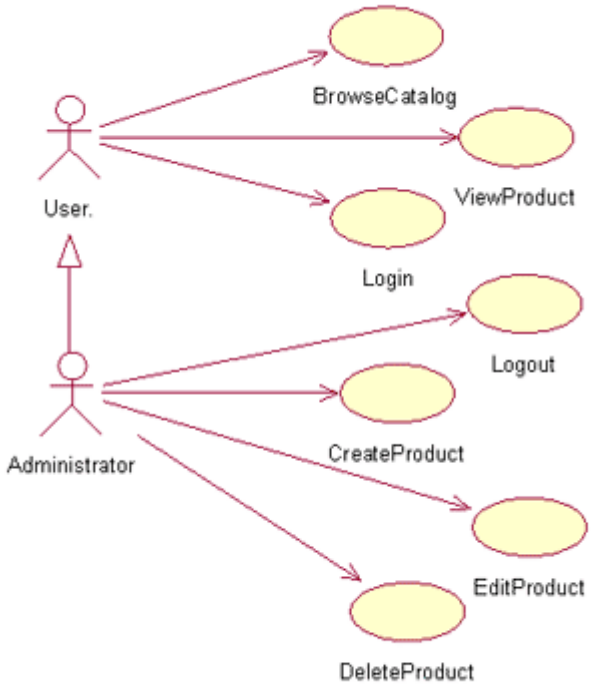


Sample use case diagram

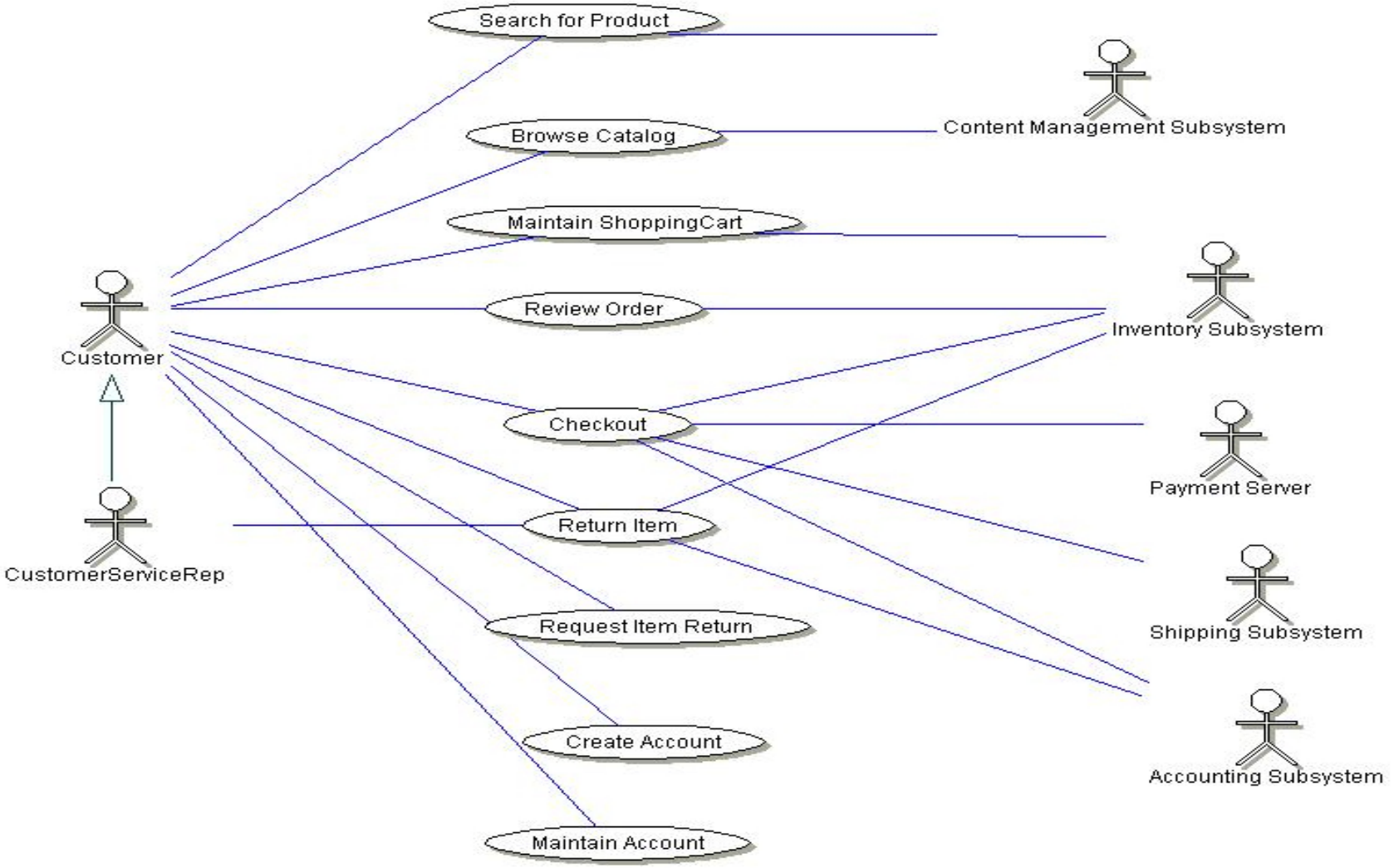


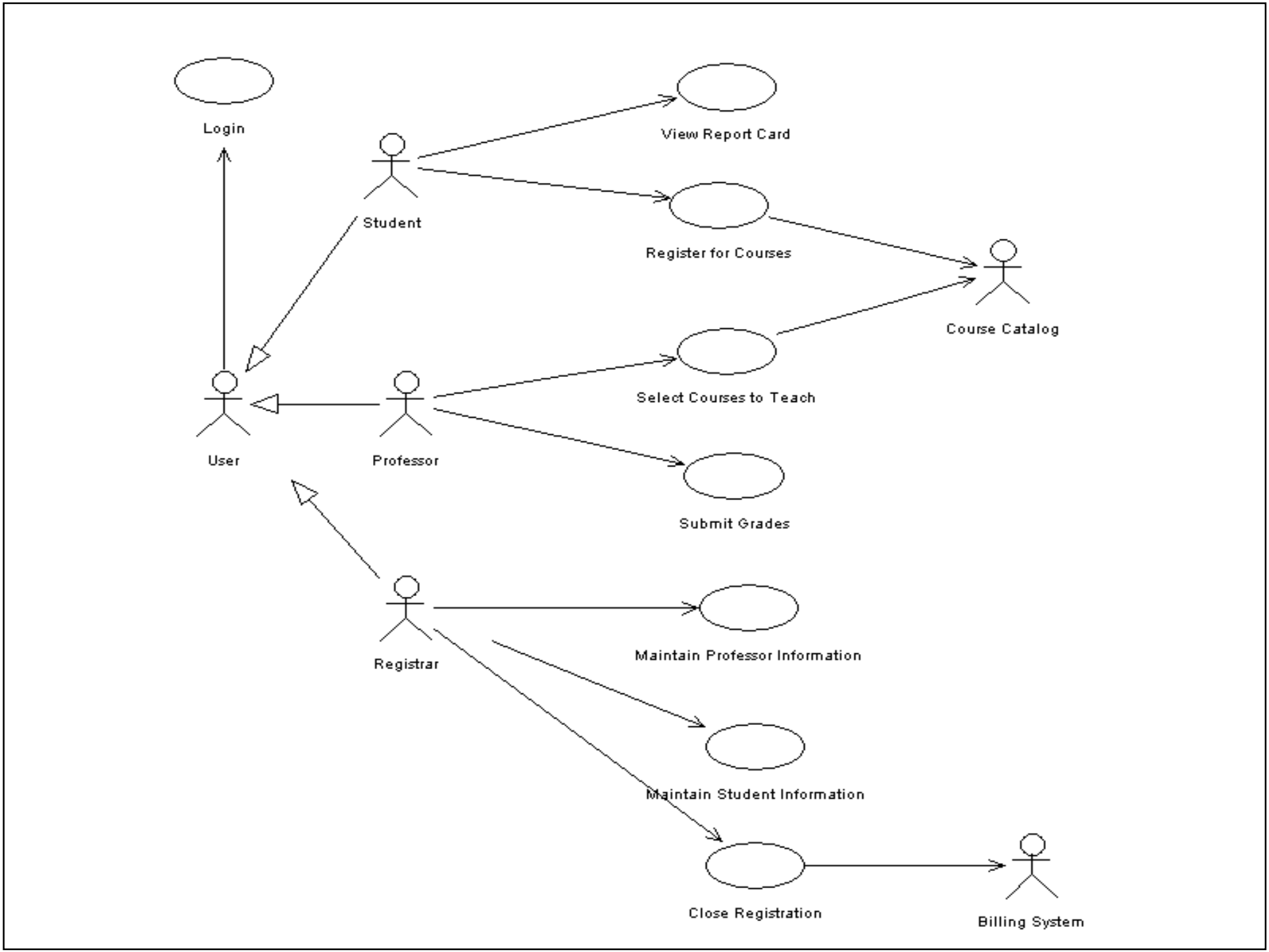

usecase - Windows Picture and Fax Viewer
http://homepages.uel.ac.uk/D.Bowden/Creating_use_case_diagram/Creating_use_case_diagram.viewlet - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://homepages.uel.ac.uk/D.Bowden/Creating_use_case_diagram/Creating_use_case_diagram.viewlet/Creating_use_case_diagram.swf
Go
Google
Bookmarks 191 blocked Check AutoLink AutoFill Send to
Web Search Bookmarks Settings Mail My Yahoo! Answers Games Anti-Spy News Shopping
http://homepages.uel.ac.uk/D.... Add Tab
File Edit View Format Browse Report Query Tools Add-Ins Window Help
test3
Use Case View
User A
Lecturer A
Student A
Add Student
Amend Student
Check Grades
Delete Student
Find Student
Associations
Logical View
Component View
Deployment View
Model Properties
Lecturer A
Amend Student
<<include>>
Find Student
Add Student
<<extend>>
<<include>>
Student A
Delete Student
Check Grades
For Help, press F1
Default Language: Analysis
NUM
http://www.yahoo.com/?fr=fptb-cclean
Internet

Sample use case diagram

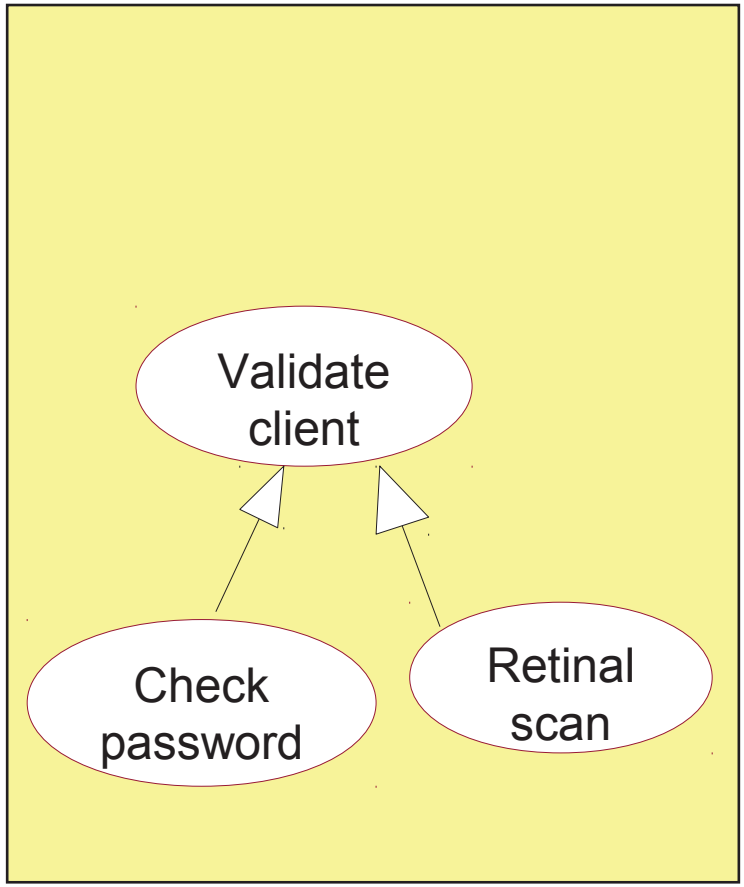


Sample use case diagram




Login
View Report Card
Student
Register for Courses
Course Catalog
Select Courses to Teach
User
Professor
Submit Grades
Registrar
Maintain Professor Information
Maintain Student Information
Close Registration
Billing System

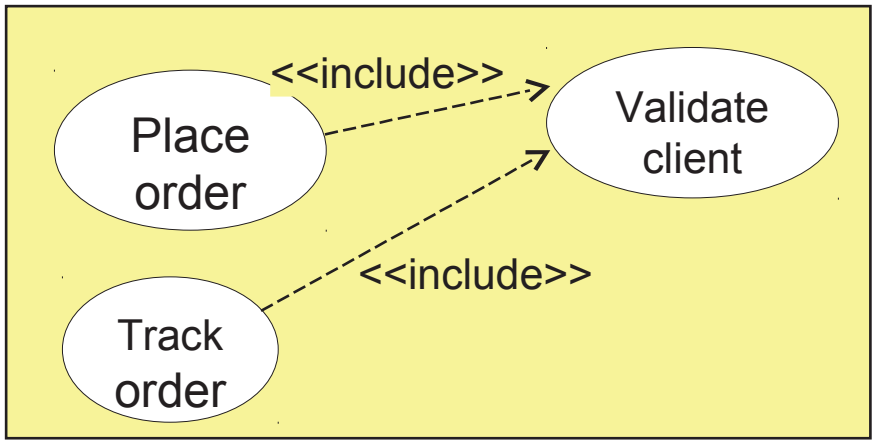
Organizing Use Cases: Generalization relationship



- child use case inherits behavior and meaning of the parent use case
- child may add or override the parent's behavior
- child may be substituted any place the parent appears

Organizing Use Cases: "Include" relationship

- Used to avoid describing the same flow of events several times, by putting the common behavior in a use case of its own
- Avoids copy & paste of parts of use case descriptions



“Include” Example

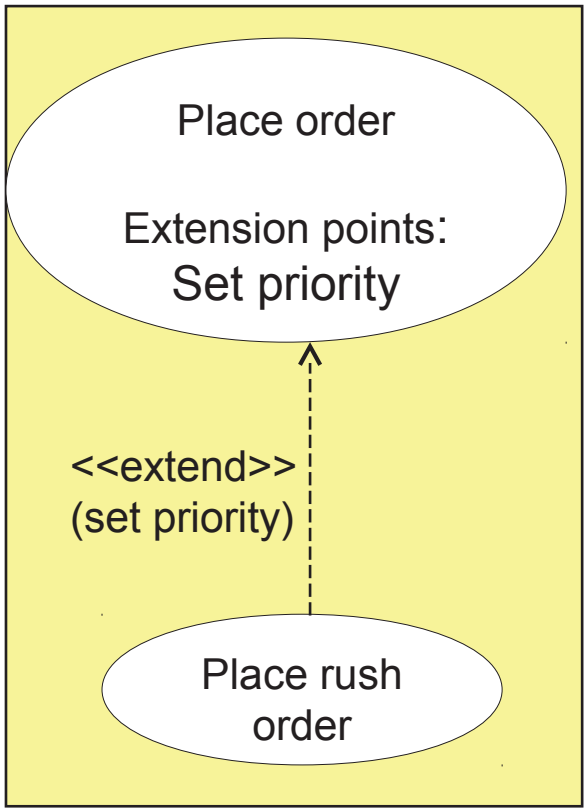
Use Case: Track order

Precondition: Financial Officer is logged in

Main flow:

1. Obtain and verify order number
2. *Include (Validate client)*
3. For each part in the order,…

Organizing Use Cases: "Extend" relationship



- Allows to model the part of a use case the user may see as optional
- Allows to model conditional subflows
- Allows to insert subflows at a certain point, governed by actor interaction

- extension points (in textual event flows)

“Extend” Example

Use Case: Place order

Precondition: Financial Officer is logged in

Main flow:

1. …
2. Collect the client’s order items
3. *(set priority)*
4. Submit order for processing

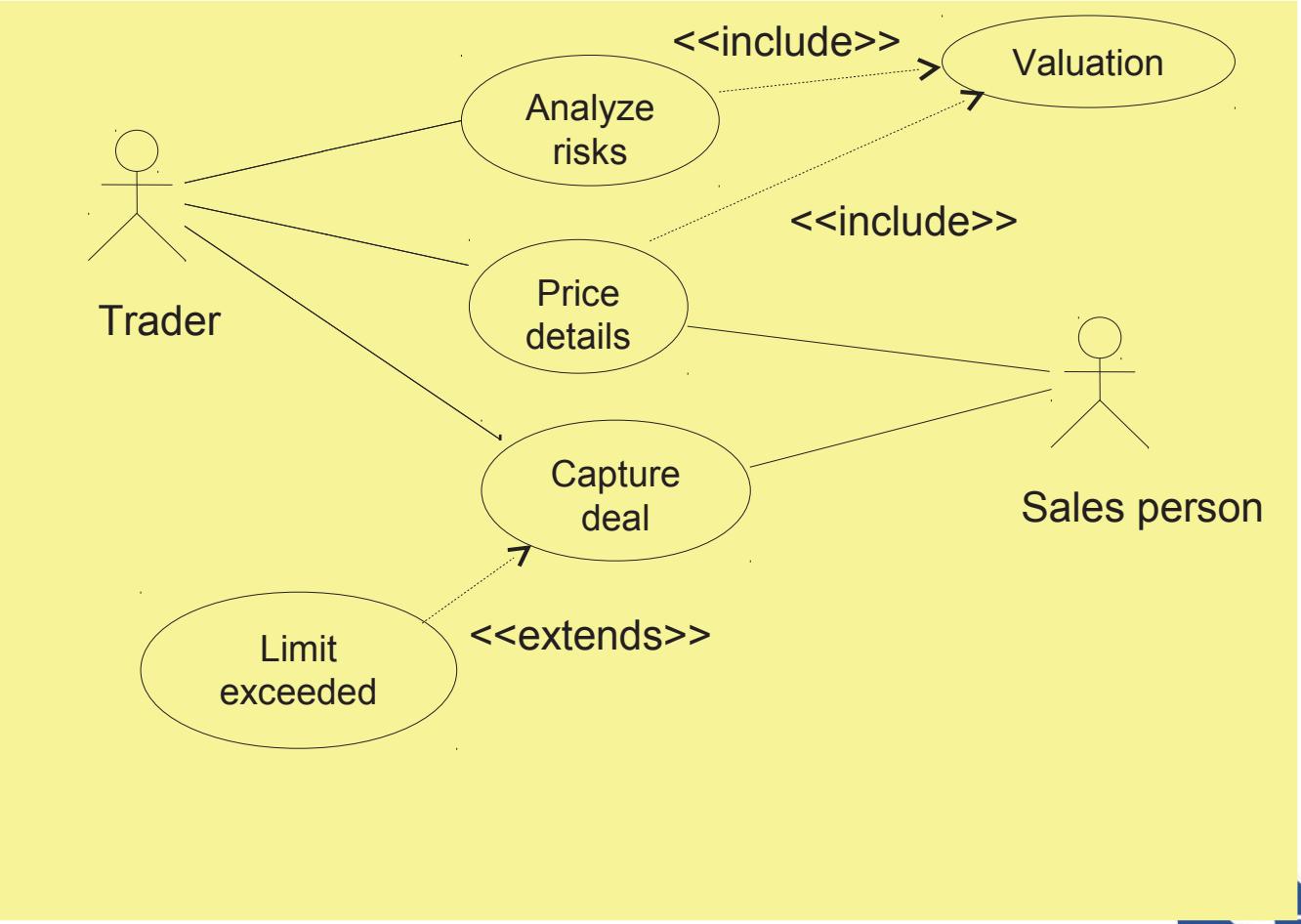
Using “Extends” relationship

- Capture the base use case
  ☞ if we later delete extension points, we still have to have a use case achieving something!
- For every step ask
  - - what could go wrong?
  - - how might this work out differently?
- Plot every variation as an extension of the use case

Comparing extends/includes

- Different intent
  - extend
    - to distinguish variants
    - associated actors perform use case and all extensions
    - actor is linked to “base” case
  - include
    - to extract common behavior
    - often no actor associated with the common use case
    - different actors for “caller” use cases possible

A use case diagram



Properties of use cases

- Granularity: fine or coarse
- Achieve a discrete goal
- Use cases describe externally required functionality
- Often: Capture user-visible function

When and how

When:

- Requirements capture - first thing to do

How:

- Use case: Every discrete thing your customer wants to do with the system
  - give it a name
  - describe it shortly (some paragraphs)
  - add details later

Recipe

- Identify actors that interact with the system
- Organize actors
- Consider primary ways of interaction with actors
- Consider exceptional ways
- Organize behaviors as use cases, using generalize/include/extend relationships
- Describe what actors require from system, not what goes on in the system!