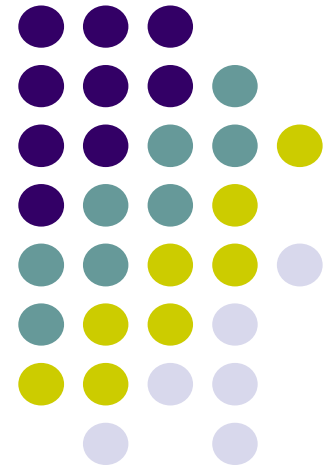


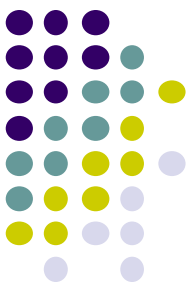
Actors and Use Cases.

Use Case Diagrams.

Relationships

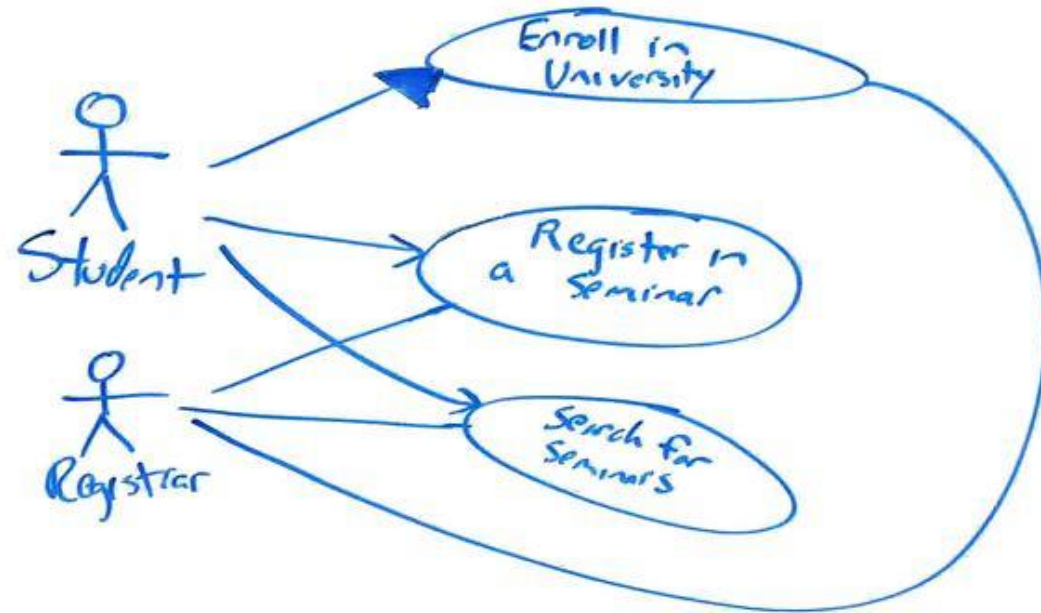
Definitions
Diagrams
Relationships
Examples
Case Study

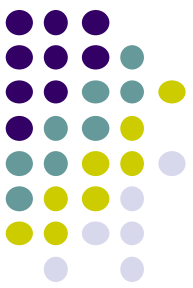




UML use case diagrams

- UML use case diagrams provide overview of usage requirements for a system.
- For actual system or software engineering use case diagrams describe actual system/software requirements
- Useful also for simple presentations to management and/or project stakeholders





History

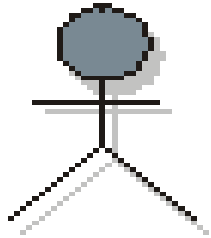
- 1986 - Ivar Jacobson formulated the textual, structural and visual modeling techniques for specifying use (originally usage) cases
- 1990 - use cases started becoming one of the most common practices for capturing functional requirements
- 1992 - Jacobson's book "Object-Oriented Software Engineering - A Use Case Driven Approach"
- 1995 - use case diagrams included into Unified Modeling Language (UML) and the Rational Unified Process (RUP)



Elements

- **Actors** - a person, organization, or external system that plays a role in one or more interactions with your system
- **Use cases** - describe a sequence of actions that provide something of measurable value to an actor
- **Associations** - exist whenever an actor is involved with an interaction described by a use case
- **Other relations** – *include, extend, generalize* and *depend*
- **System boundary boxes** (optional) - rectangles around the use cases to indicate the scope of your system
- **Packages** (optional) - UML constructs that enable you to organize model elements (such as use cases) into groups.

Defining Actors



Actor

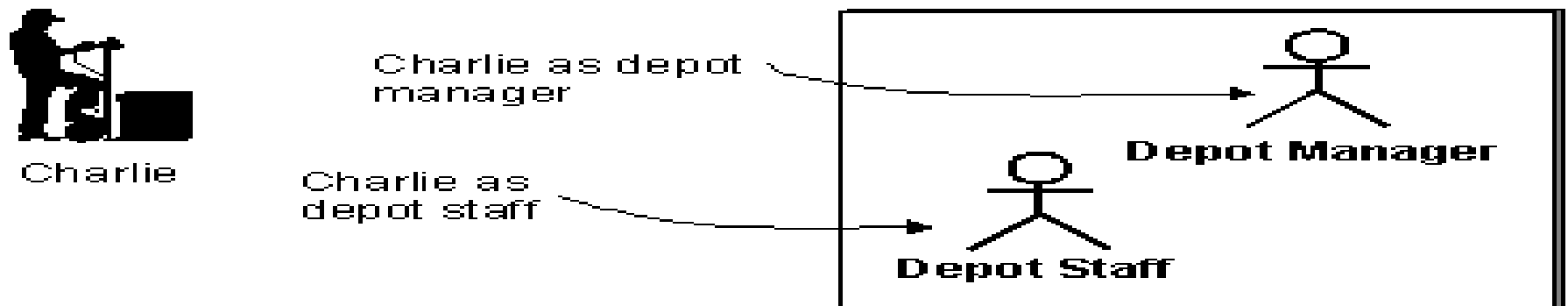
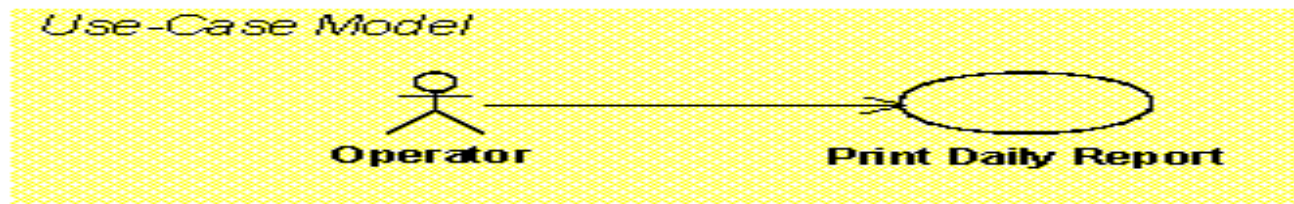
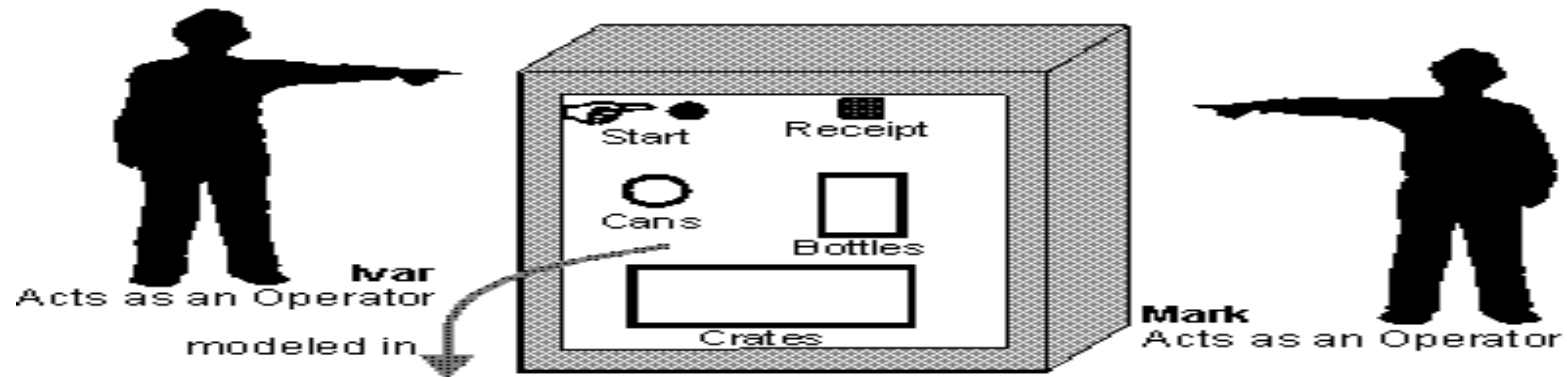
An **actor instance** is *someone* or *something* outside the system that interacts with the system.

An **actor class** defines a set of actor instances, in which each actor instance plays the same role in relation to the system.

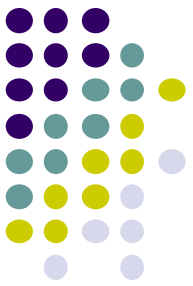
To fully understand the system's purpose you must know **who** the system is for, or who will use the system. Different user types are represented as actors.

An actor is anything that exchanges data with the system. An actor can be a user, external hardware, or another system

Differences between an Actor and an Individual System User



How to Find Actors



- Who will supply/use/remove information?
- Who will use this functionality?
- Who is interested in any requirement?
- Where in the organization is the system used?
- Who will support/maintain the system?
- What are the system's external resources?
- What other systems will need to interact with this one?

Actors as Different Aspects of System's Surroundings



- Users who execute the system's main functions – for a Depot-Handling System: Depot Staff and Order Registry Clerk.
- Users who execute the system's secondary functions, such as system administration - Depot Manager.
- External hardware the system uses - a ventilation system that controls the temperature in a building continuously gets metered data from sensors in the building. Sensor is therefore an actor.
- Other systems interacting with the system - An ATM (Automated Teller Machine) must communicate with the central system that holds the bank accounts. The central system is probably an external one, and should therefore be an actor.

Actors Help Define System Boundaries



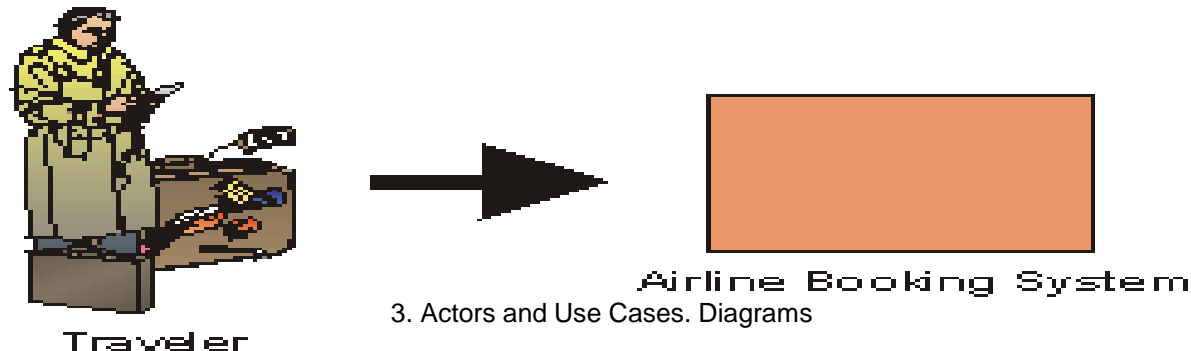
Only those who ***directly communicate with the system*** need to be considered as ***actors***. Otherwise, you are attempting to model the business in which the system will be used, not the system itself.

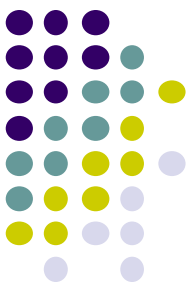
In an airline booking system, what would the actor be?

1. If the system is to be used by a travel agent, the actor would be travel agent.



2. When users to connect via the Internet:





Documenting Actor Characteristics

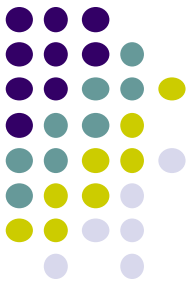
Brief description:

- ◆ What or who the actor represents?
- ◆ Why the actor is needed?
- ◆ What interests the actor has in the system?

Actor characteristics might influence how the system is developed:

- The actor's scope of responsibility.
- The physical environment in which the actor will be using the system.
- The number of users represented by this actor.
- Others.

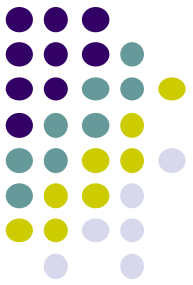
Defining Use Cases



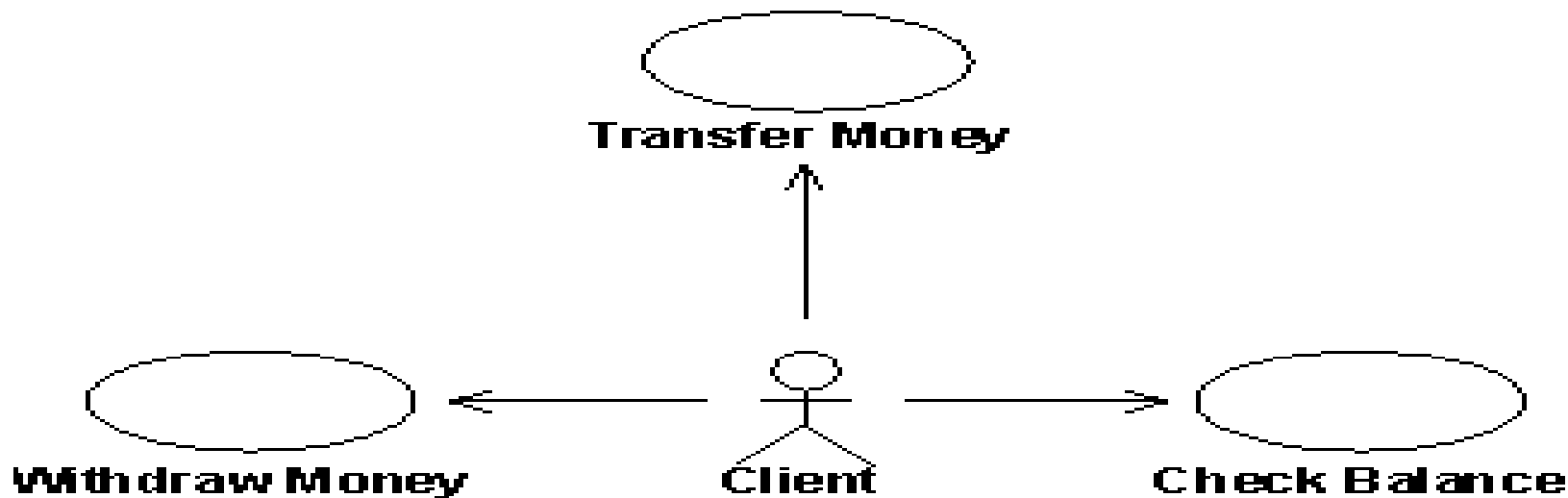
Use Case

A **use-case instance (scenario)** is a sequence of actions a system performs that yields an observable result of value for one or more particular actors or other stakeholders of the system.

A **use-case** defines a set of use-case instances.

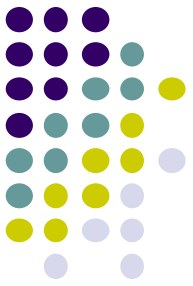


Use Case Example



An ATM example - the system functionality is defined by different use cases, each of which represents a specific flow of events, defines what happens in the system when the use case is performed, and has a task of its own to perform.

How to Find Use Cases

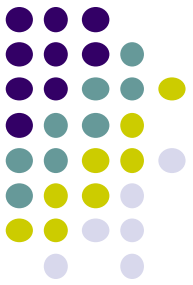


- What are the system tasks for each actor you have identified?
- Does the actor need to be informed about certain occurrences in the system?
- Will the actor need to inform the system about sudden, external changes?
- Does the system supply the business with the correct behavior?
- Can all features be performed by the use cases you have identified?
- What use cases will support and maintain the system?
- What information must be modified or created in the system?

Use cases types:

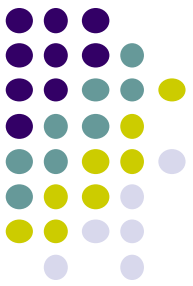
- System start and stop.
- Maintenance of the system (add user, ...).
- Maintenance of data stored in the system.
- Functionality needed to modify behavior in the system.

Use Case Documenting – Flow of Events

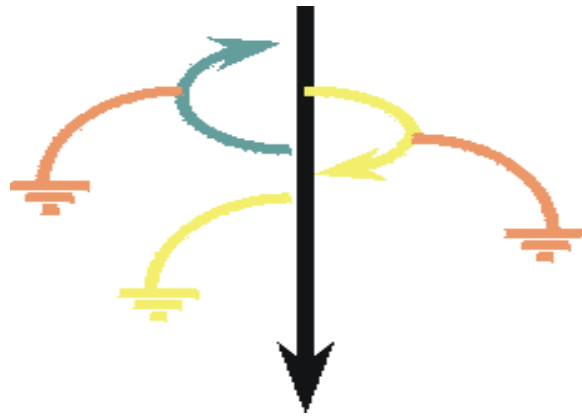


The **Flow of Events** of a use case contains the most important information derived from use-case modeling work. Its contents:

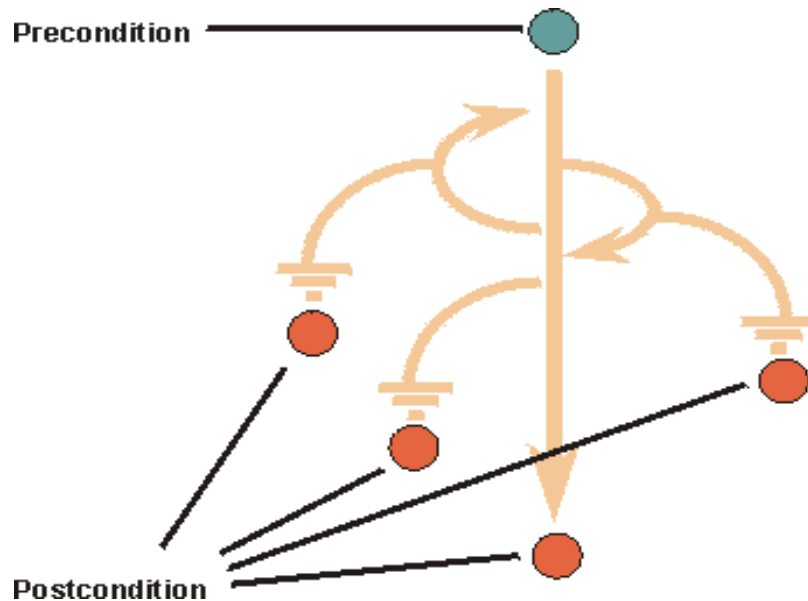
- Describe how the use case starts and ends
- Describe what data is exchanged between the actor and the use case
- Do not describe the details of the user interface, unless it is necessary to understand the behavior of the system
- Describe the flow of events, not only the functionality. To enforce this, start every action with "When the actor ... "
- Describe only the events that belong to the use case, and not what happens in other use cases or outside of the system
- Avoid vague terminology such as "for example", "etc. " and "information" - description stile
- Detail the flow of events - all "*whats*" should be answered.



Flow of Events - Structure

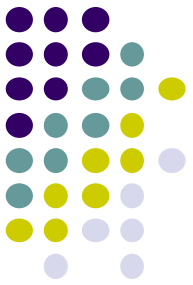


Basic (the straight arrow) and **alternative** flows of events (the curves).



A **precondition** is the state of the system and its surroundings that is required before the use case can be started. A **postcondition** is the states the system can be in after the use case has ended.

How to describe a use case scenario?



*While a **use case** is an abstraction that describes all possible scenarios, a **scenario** is an use case instance with concrete set of actions. A three fields template [Bruegge&Dutoit, 2004]:*

- Name – unambiguous, underlined;
- Participating actors – underlined names;
- Flow of events – sequence of numbered interactions for the use case; accomplished either by the actor (left column) or by the system (right)

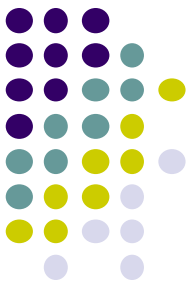
There are no Entry and Exit conditions – as they are abstractions to describe a range of conditions under which a use case is invoked.



How to describe a use case?

A six fields template [Bruegge&Dutoit, 2004]:

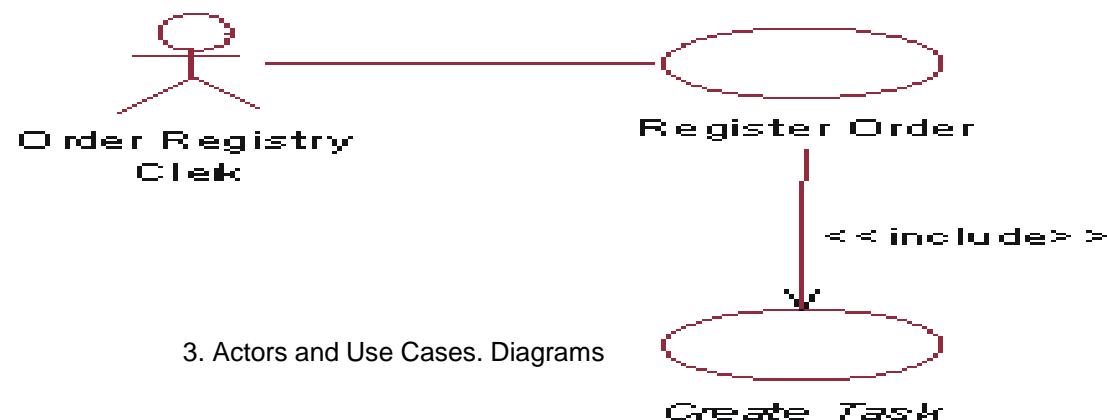
- Name – unambiguous, unique across the system;
- Participating actors;
- Entry conditions – need to be TRUE before use case initiation;
- Flow of events:
 - sequence of numbered interactions for the use case;
 - accomplished either by the actor (left column) or by the system (right)
- Exit conditions - need to be TRUE after use case completion;
- Quality conditions – non-functional requirements...



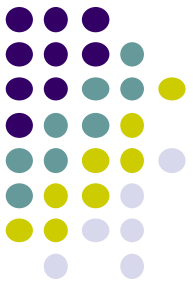
Concrete and Abstract Use Cases

A **concrete** use case is initiated by an actor and constitutes a complete flow of events (instance of the use case performs the entire operation called for by the actor).

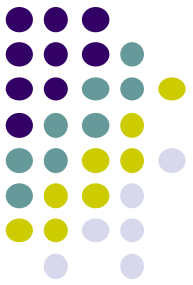
An ***abstract*** use case (written in *italics*) is never instantiated in itself. Abstract use cases are *included in, extended into, or generalizing* other use cases. When a concrete use case is initiated, an instance of the use case is created. This instance also exhibits the behavior specified by its associated *abstract* use cases. Thus, no separate instances are created from *abstract* use cases.



ID, Rank, Leaf and Root use cases



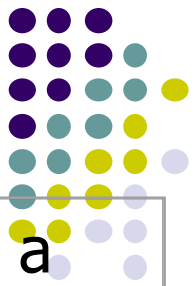
ID	A unique value for identifying the use case.
Rank	Describe the importance of the use case. The higher the ranking implies that more attention is needed.
Leaf	Indicates whether it is possible to further specialize an use case. If the value is true, then it is not possible to further specialize the use case.
Root	Indicates whether the use case has no ancestors (true for no ancestors).



Associations (relationships)

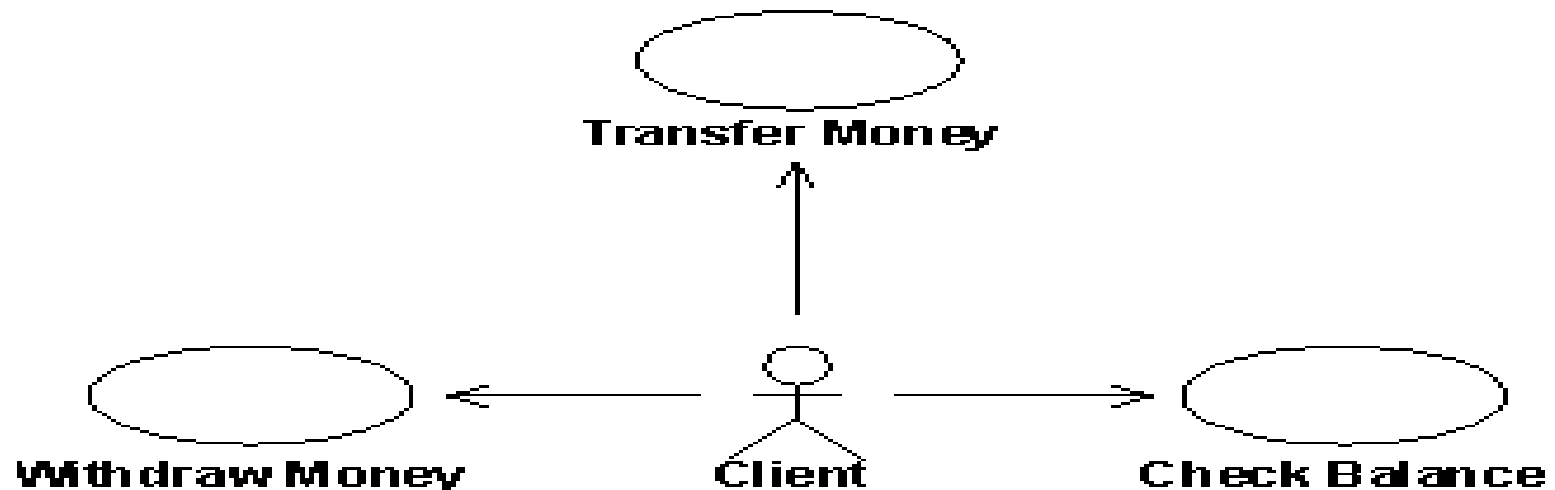
- Associations between actors and/or use cases are indicated in use case diagrams by solid lines.
- An association exists whenever an actor is involved with an interaction described by a use case.
- Associations are modeled as lines connecting use cases and actors to one another
- The arrowhead is often used to indicate the direction of the initial invocation of the relationship

Communicate-Relationship



An **communicate-relationship** is a relationship between a use case and an actor or b/n use cases and represents information exchange; used to define user functionality.

«communicate»

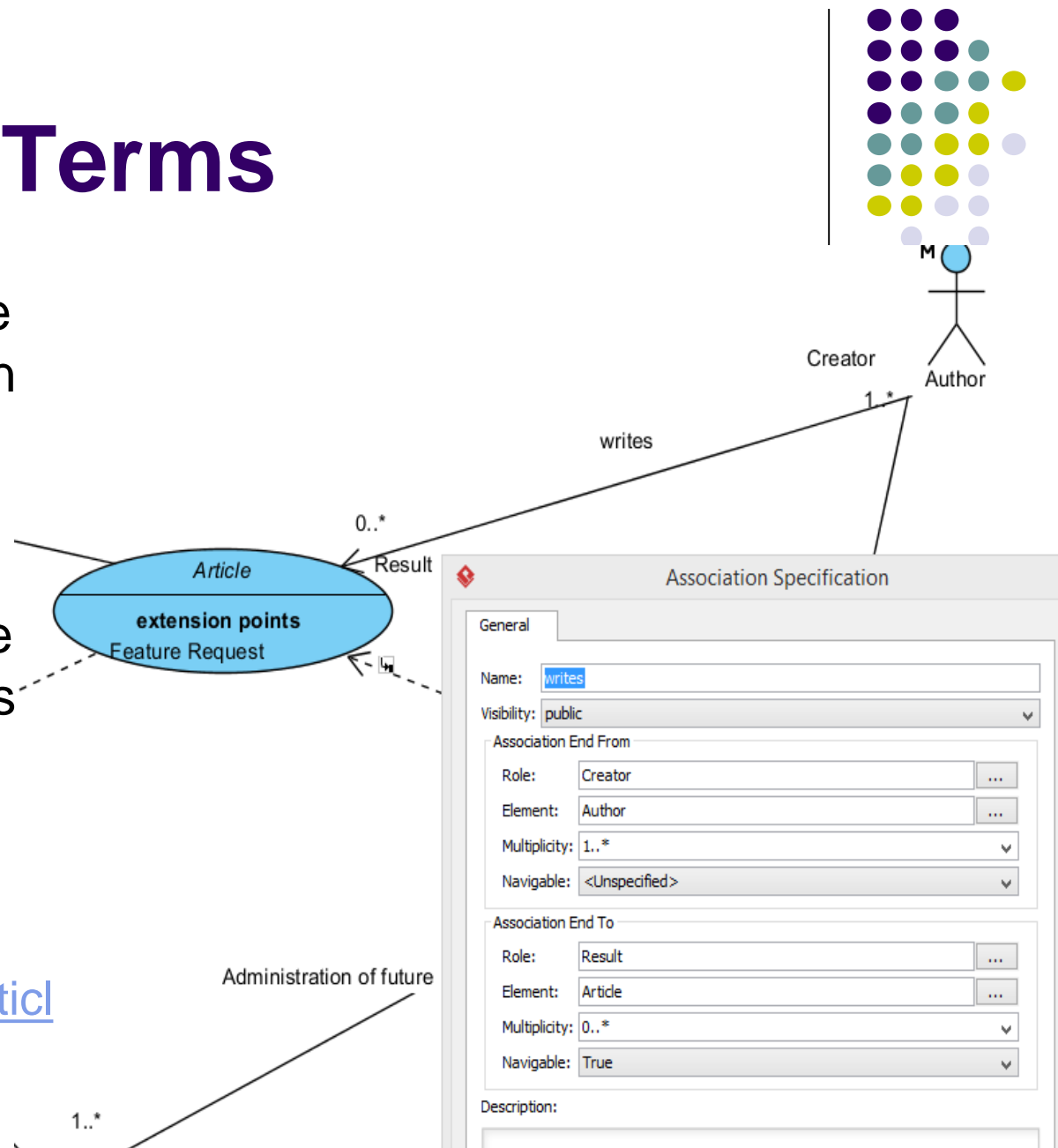


UML 2.0 New Terms

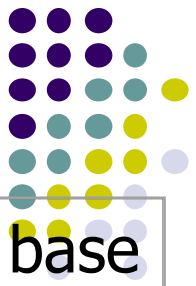
- Use Case **Multiplicities** lie on the association between actors and use cases.
- The definition of **multiplicities** in the use case diagram is exactly the same as they are in a class diagram.

* What's New in UML 2? The Use Case Diagram—by Randy Miller, June 30, 2003

<http://community.borland.com/article/0,1410,30166,00.html>

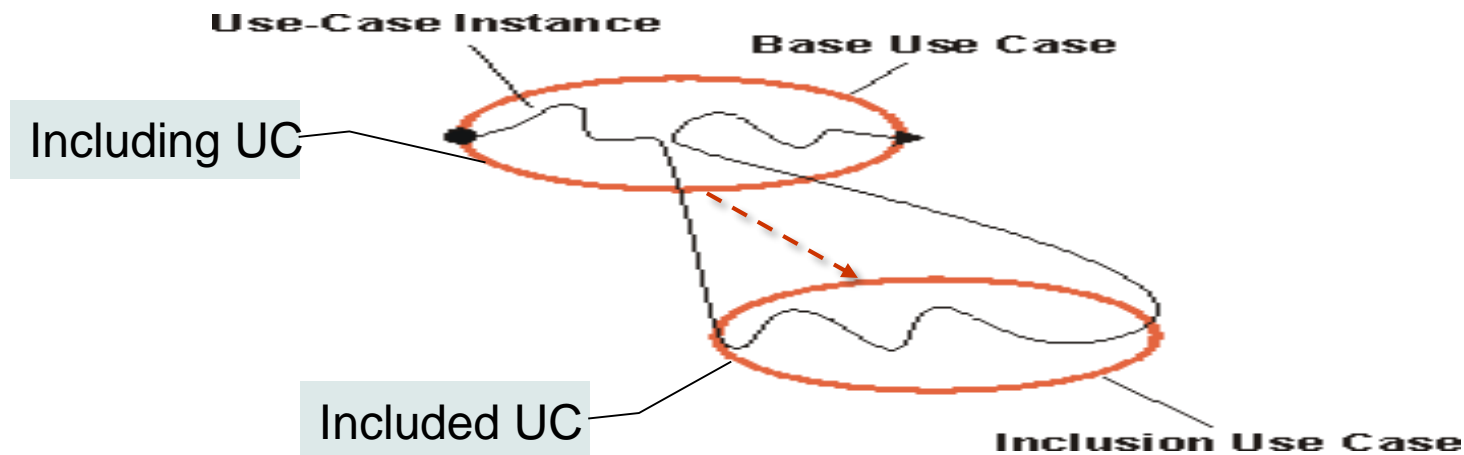


Include-Relationship



An **include-relationship** is a directed relationship from a base use case to an inclusion use case, specifying how the behavior defined for the inclusion use case is non-optionally, explicitly inserted into the behavior defined for the base use case.

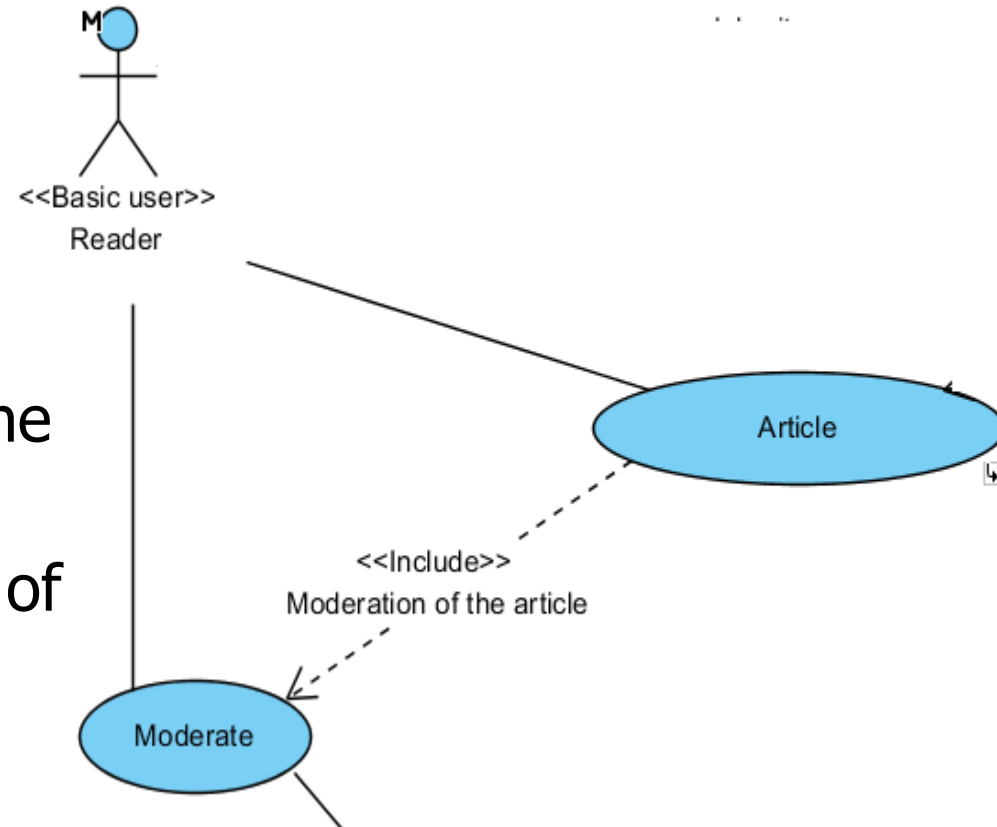
«include»

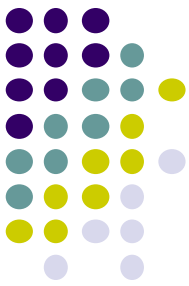


Executing a use-case instance following the description of a base use case including its inclusion.

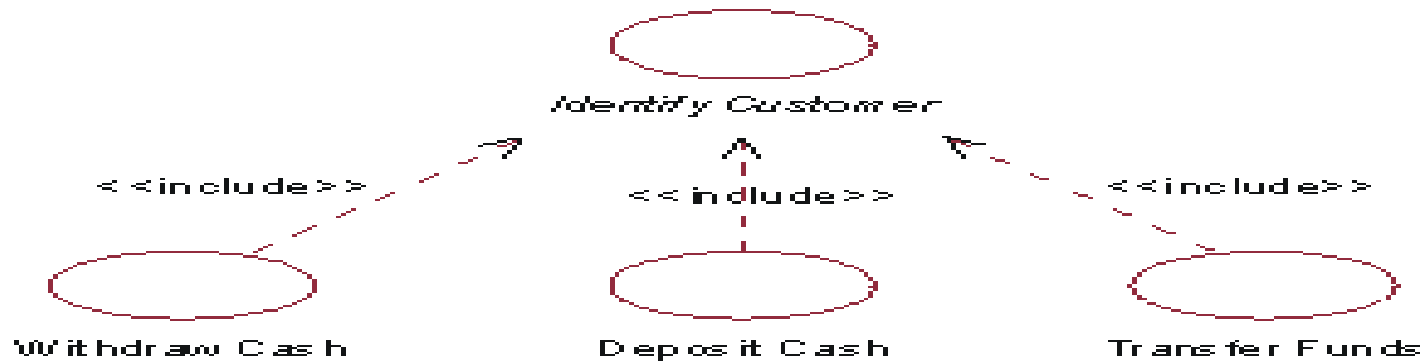
More about <<Include>>

- Including use case includes the “addition” and owns the include relationship.
- Addition is use case that is to be included.
- The including use case may only depend on the result (value) of the included use case.
- This value is obtained as a result of the execution of the included use case.





Example of Includes



In the ATM system, the use cases Withdraw Cash, Deposit Cash, and Transfer Funds all include the use case Identify Customer.

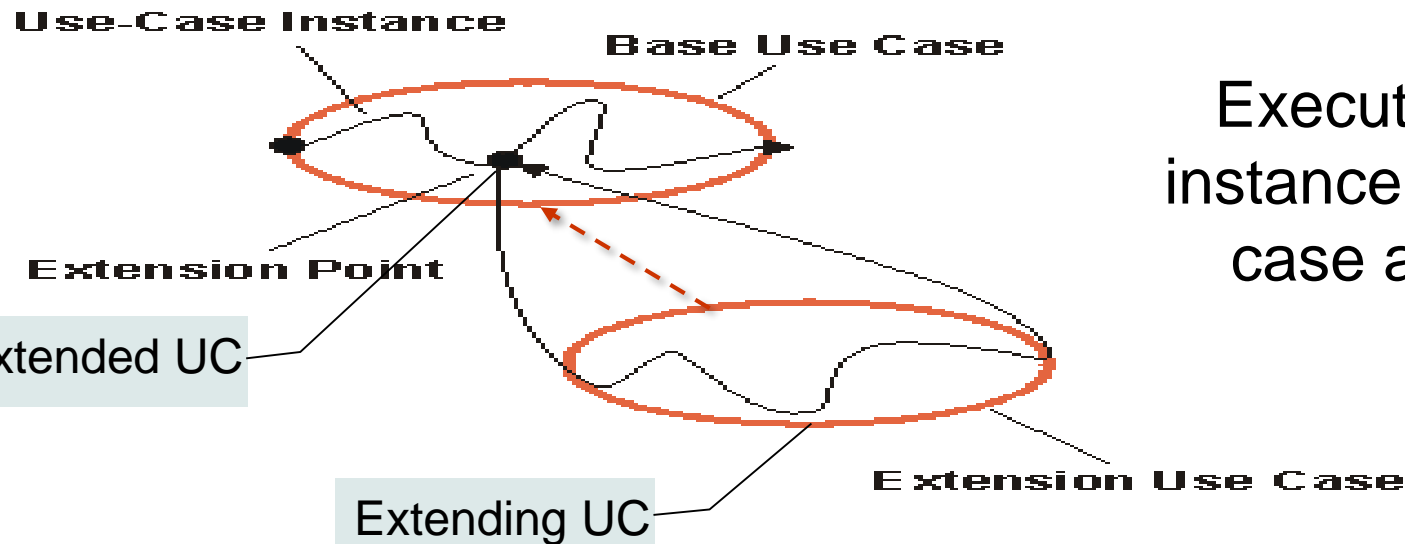
- The base use case has control of the relationship to the inclusion and can depend on the result of performing the inclusion,
- but neither the base nor the inclusion use case may access each other's attributes.
- The inclusion is in this sense encapsulated, and represents behavior that can be reused in different base use cases.



Extend-Relationship

An **extend-relationship** goes from an extension use case to a base use case, specifying how the behavior defined for the extension use case can be inserted into the behavior of the base use case. It is implicitly inserted in the sense that the extension is not shown in the base use case.

«extend»

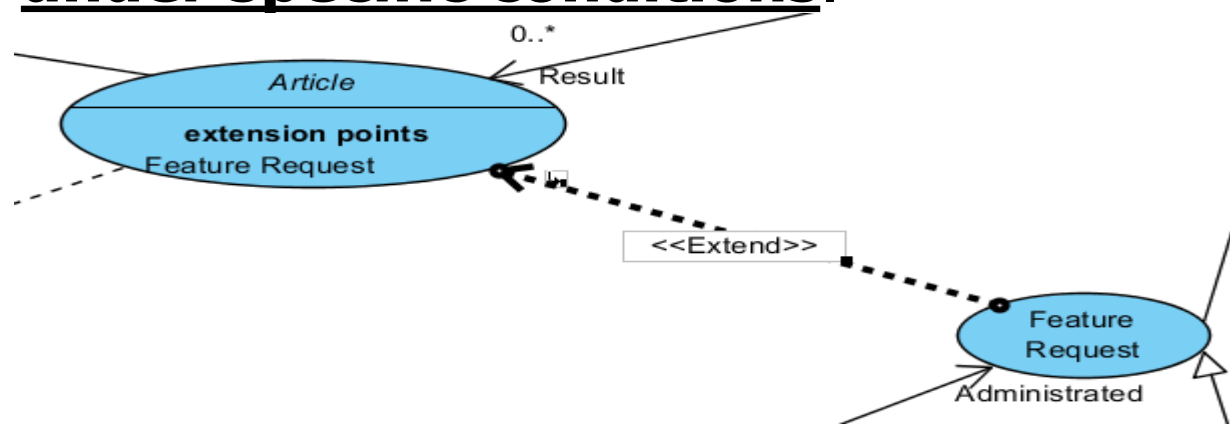


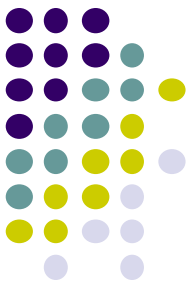
Execution of a use-case instance follows a base use case and its extension.

More about “Extend”



- This relationship specifies that the behavior of a use case may be extended by the behavior of another (supplementary) use case.
- The extended use case is defined independently of the extending use case and is meaningful independently of the extending use case.
- On the other hand, the extending use case typically defines behavior that may not necessarily be meaningful by itself. Instead, the extending use case defines a set of modular behavior increments that augment an execution of the extended use case **under specific conditions**.

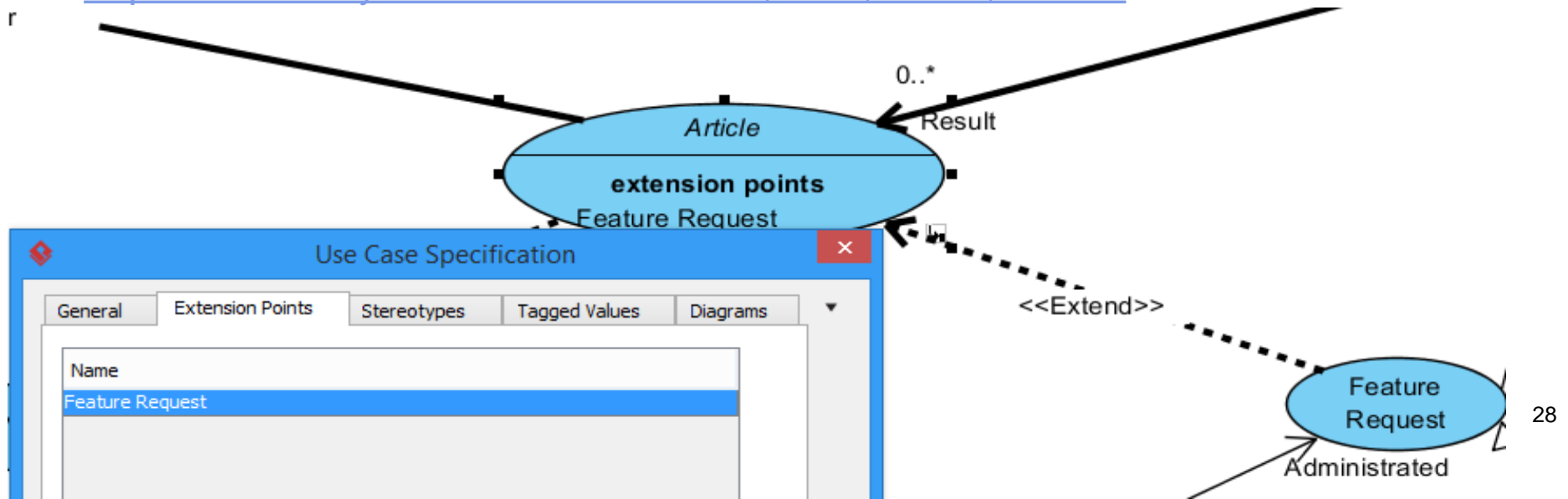




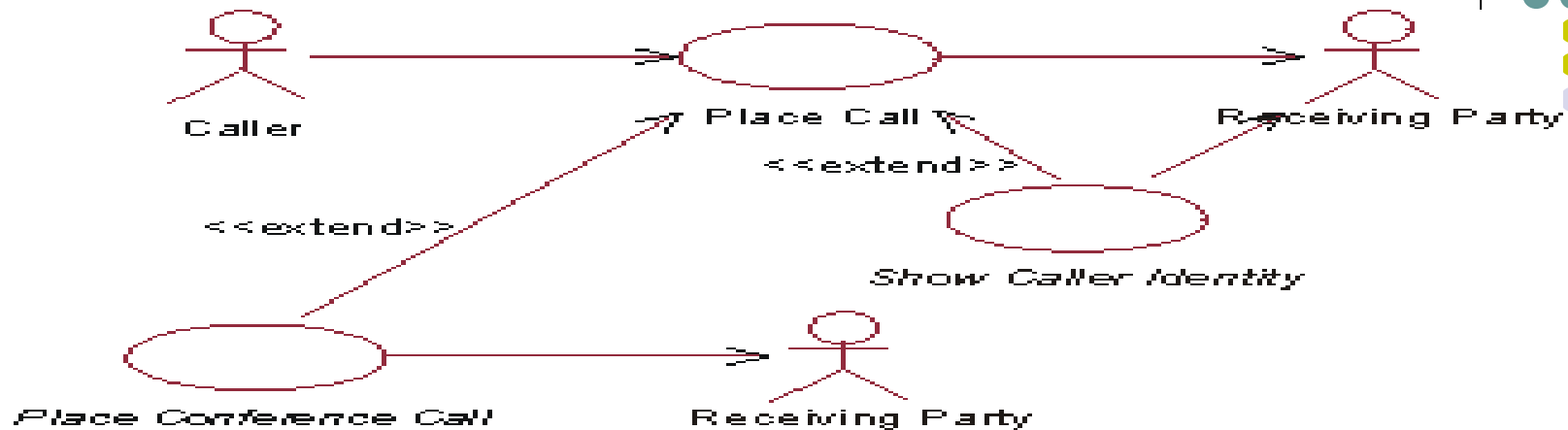
UML 2.0 New Terms

- **Extension Points** - conditions from UML 2.0 take extension points one step further. They show the actual logic necessary for one use case to extend another. They also show the exact extension point that is used between the two use cases.

* What's New in UML 2? The Use Case Diagram—by Randy Miller, June 30, 2003
<http://community.borland.com/article/0,1410,30166,00.html>



Example of Extensions

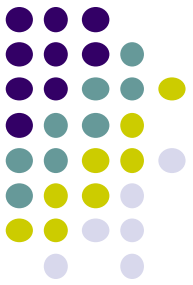


In a phone system, the primary user service is represented by the use case Place Call. Examples of optional services (extensions) are:

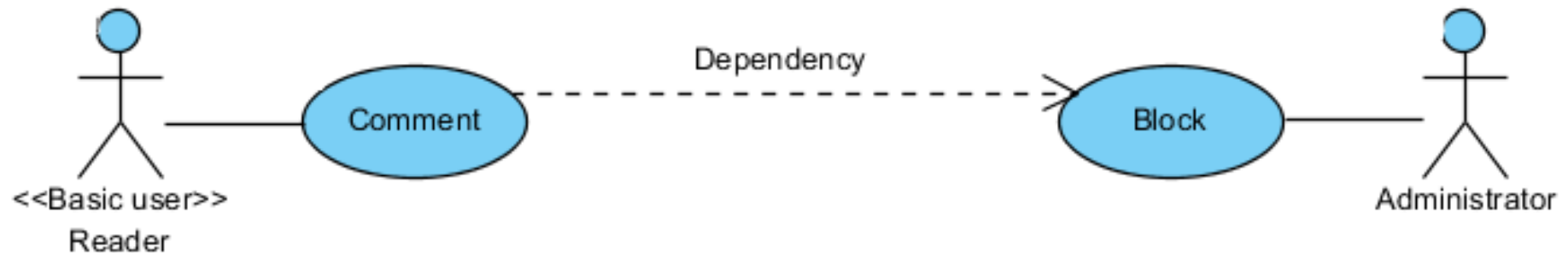
- To be able to add a third party to a call (Place Conference Call).
- To allow the receiving party to see the identity of the caller (Show Caller Identity).

The extension is conditional - it is dependent on what has happened while executing the base use case. The base use case does not control the conditions for executing the extension – the conditions are described within the extend-relationship.

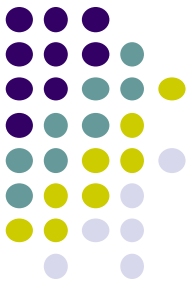
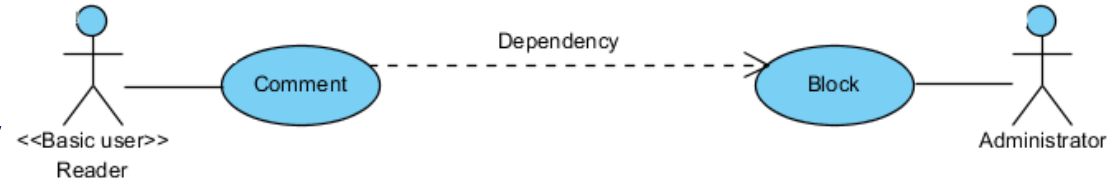
Dependency



- A dependency is a relationship that signifies that a single element or a set of model elements requires other model elements for their specification or implementation
- Changes in “Block” will affect “Comment”
- The complete semantics of the depending element (the client) is either semantically or structurally dependent on the definition of the supplier element(s)



More about dependency



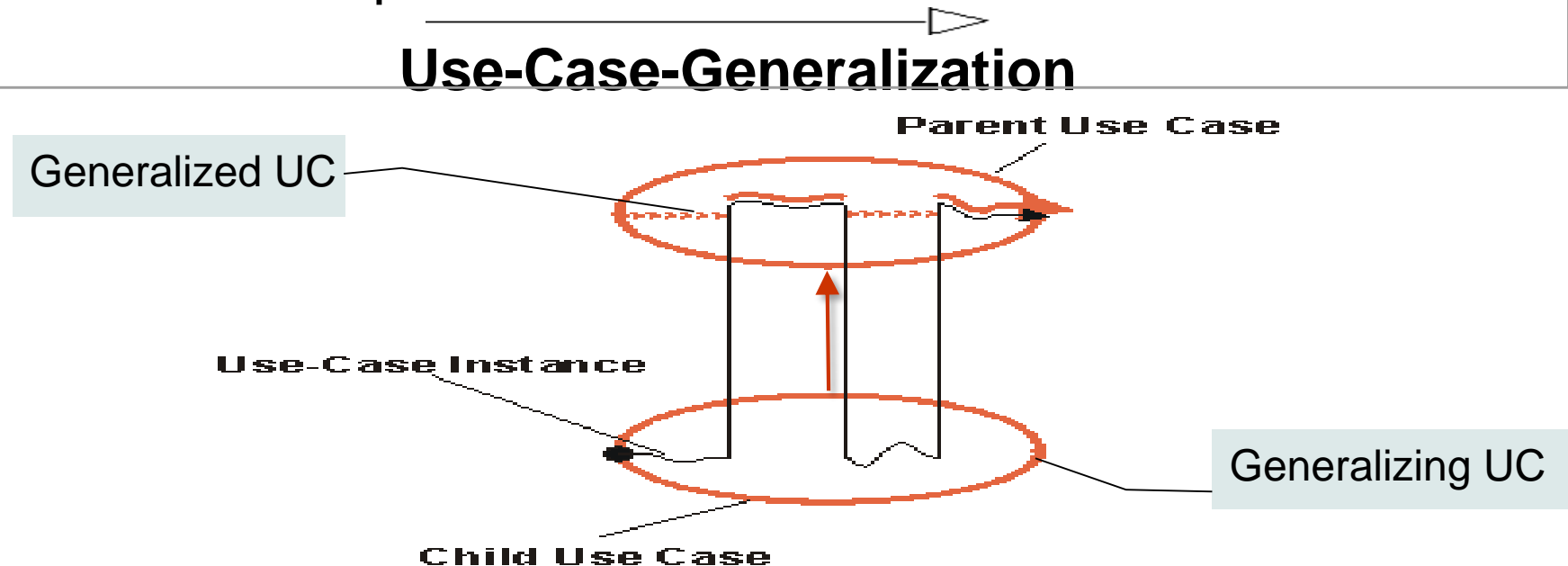
Supplier The element(s) independent of the client element(s), in the same respect and the same dependency relationship. In some directed dependency relationships (such as Refinement Abstractions), a common convention is to put the more abstract element in this role. However, we can make a more abstract element dependent on that which is more specific.

Client The element(s) dependent on the supplier element(s). In some cases (such as a Trace Abstraction) the assignment of direction (that is, the designation of the client element) is at the discretion of the modeler, and is a stipulation.



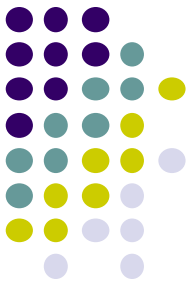
Use-Case-Generalization

A **use-case-generalization** is a taxonomic relationship from a child use case to a more general, parent use case, specifying how a child can specialize all behavior and characteristics described for the parent.

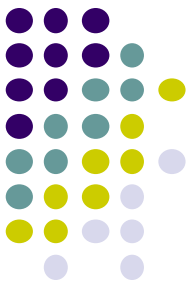


Execution: the use-case instance follows the parent use case, with behavior inserted or modified as described in the child use case.

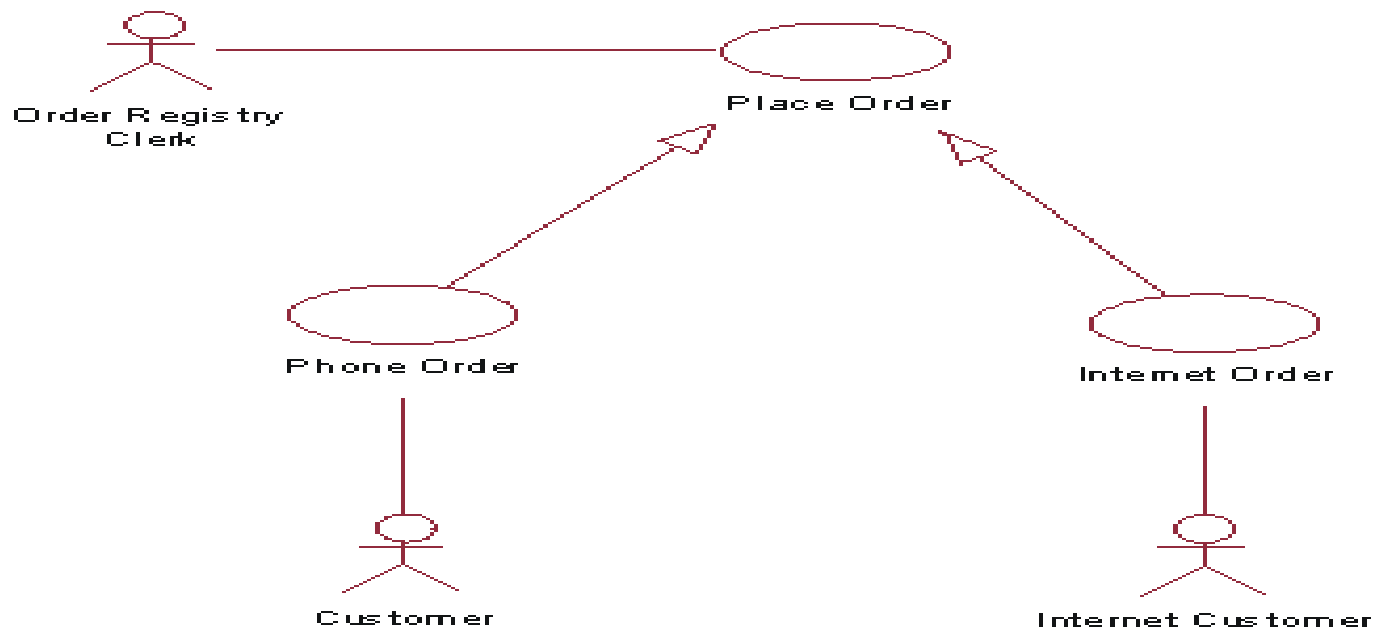
More about Generalization



General	References the general classifier in the Generalization relationship.
Specific	References the specializing classifier in the Generalization relationship.
Substitutable	Indicates whether the specific classifier can be used wherever the general classifier can be used. If true, the execution traces of the specific classifier will be a superset of the execution traces of the general classifier.



Example of Use-Case-Generalization



The actor Order Registry Clerk can instantiate the general use case Place Order. Place Order can also be specialized by the use cases Phone Order or Internet Order.

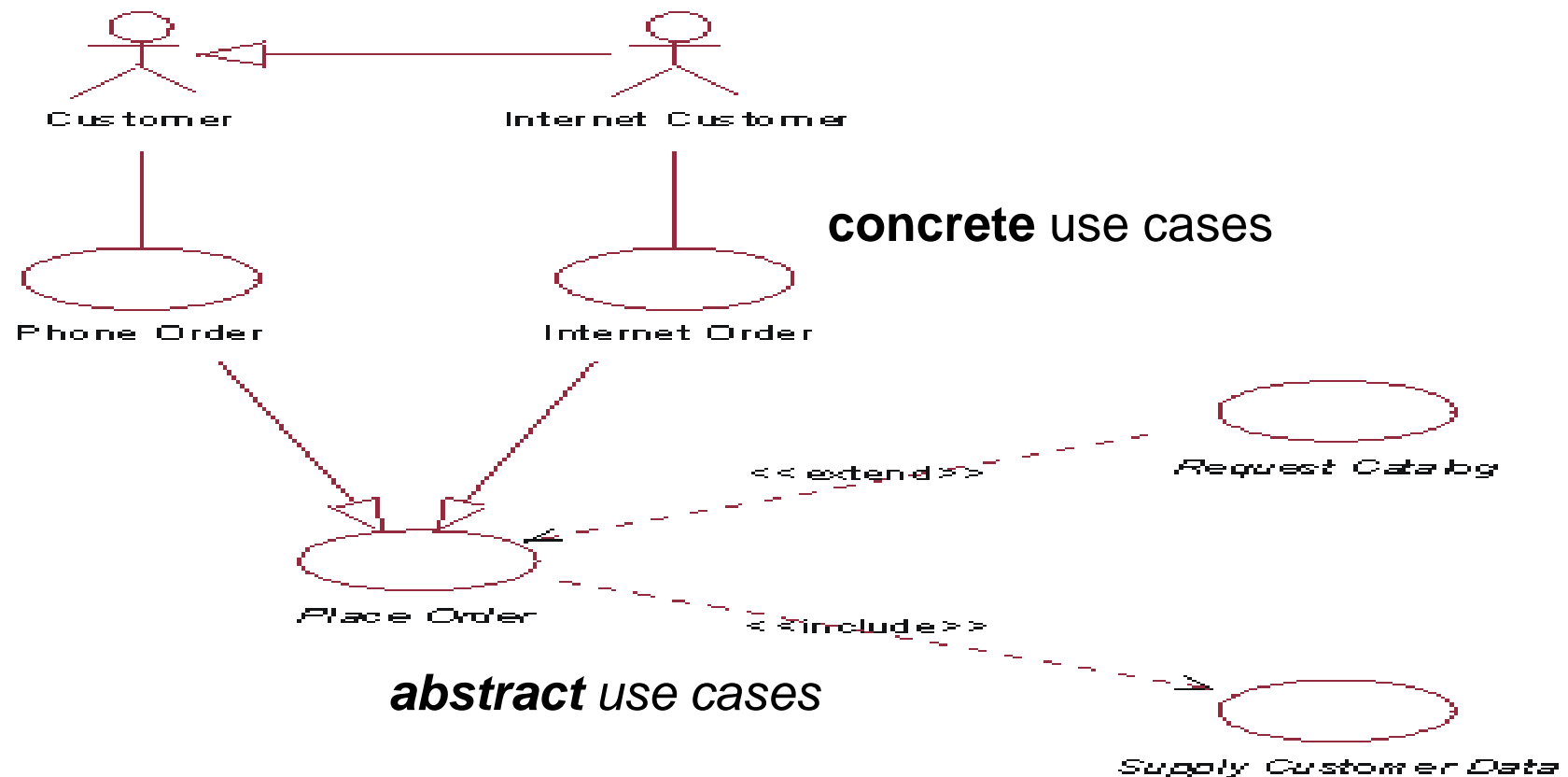
The child may modify behavior segments inherited from the parent. The structure of the parent use case is preserved by the child. Both use-case-generalization and include can be used to reuse behavior among use cases.

Use-Case Model of an Order Management System

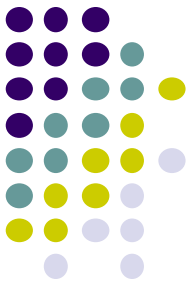


Use-Case Model

The **use-case model** is a model that describes a system's requirements in terms of use cases.



Collaboration



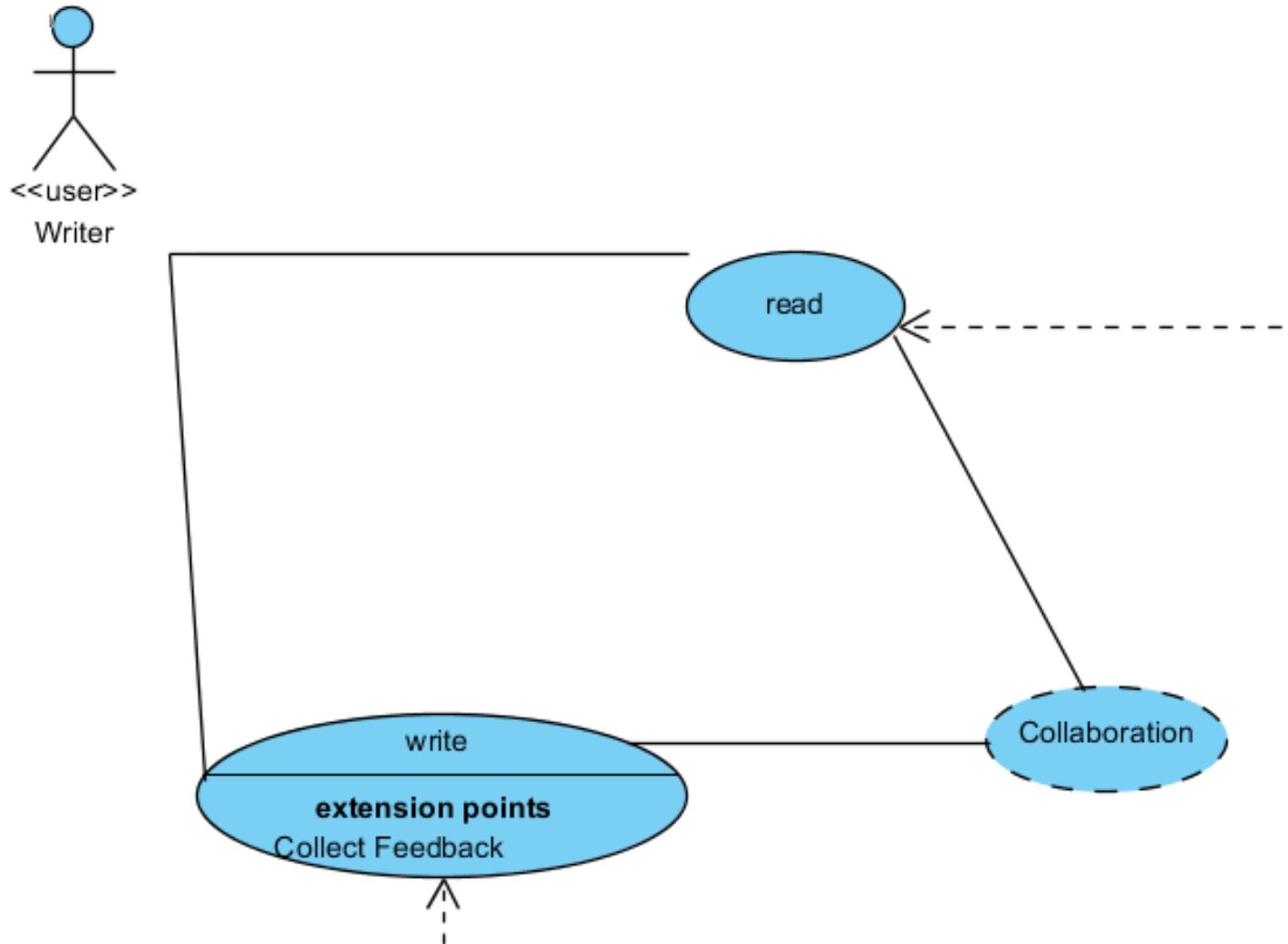
- A collaboration describes a container of collaborating elements (roles), each performing a specialized function, which collectively accomplish some desired functionality.
- Its primary purpose is to explain how a system works and, therefore, it typically only incorporates those aspects of reality that are deemed relevant to the explanation. Thus, details, such as the identity or precise class of the actual participating instances are suppressed.

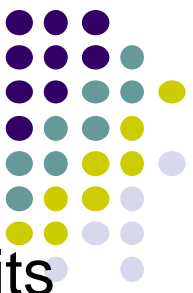
Source: UML Superstructure Specification version 2.4.1, page 174



UML collaboration

Collaboration example

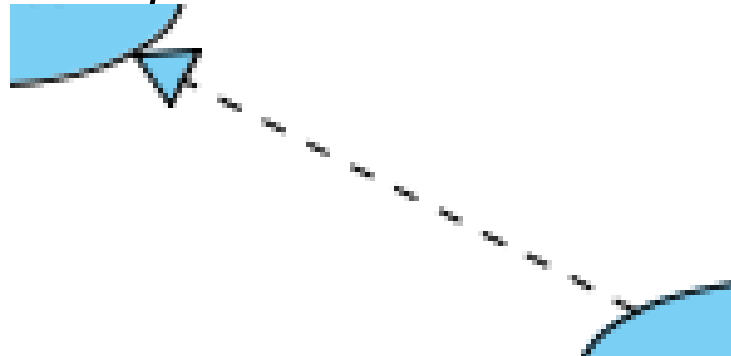




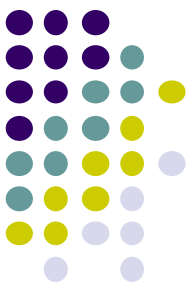
Use case realization

- A realization is a relationship between a specification and its implementation
- Realization is a specialized abstraction relationship between two sets of model elements, one representing a specification (the supplier) and the other represents its implementation (the client).
- Realization can be used to model stepwise refinement, optimizations, transformations, templates, model synthesis, framework composition, etc.

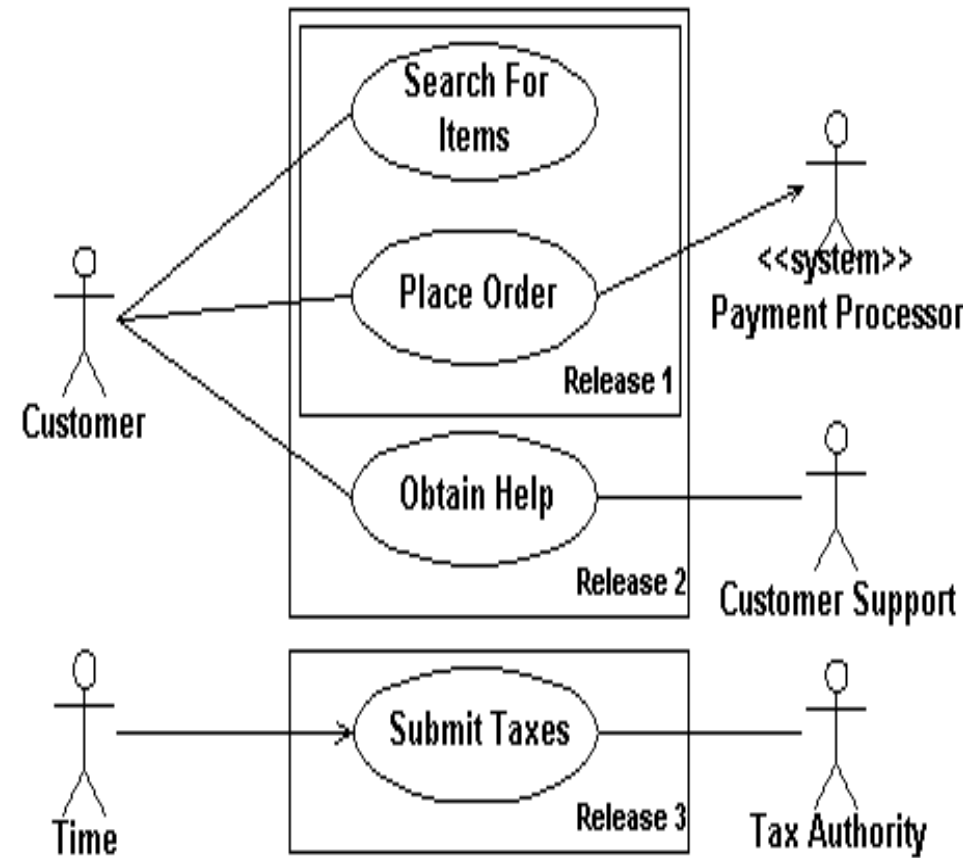
Source: UML Superstructure Specification version 2.4.1, page 131



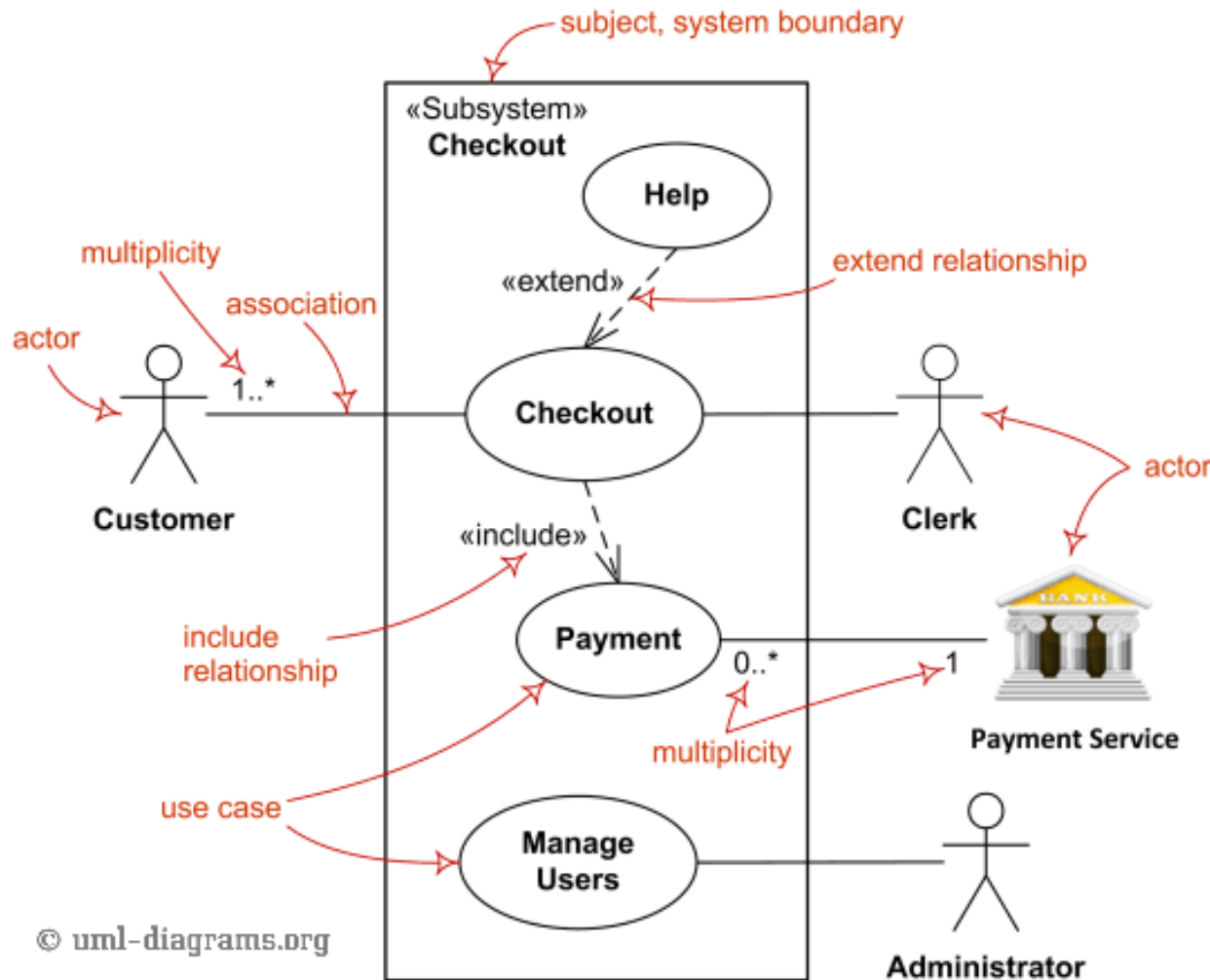
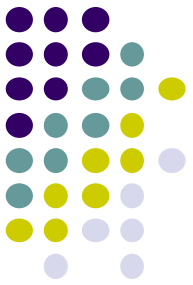
System boundary boxes



- **System boundary box (optional)** - a rectangle around the use cases to indicate the scope of your sub-system
- Anything within the box represents functionality that is in scope and anything outside the box is not
- Rarely used – i.e., to identify which use cases will be delivered in each major release of a system

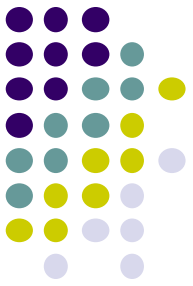


System use case diagram example



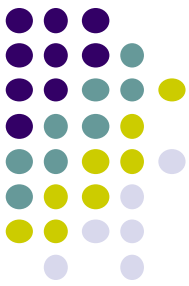
Source: <http://www.uml-diagrams.org/use-case-diagrams.html>

Business use case diagrams

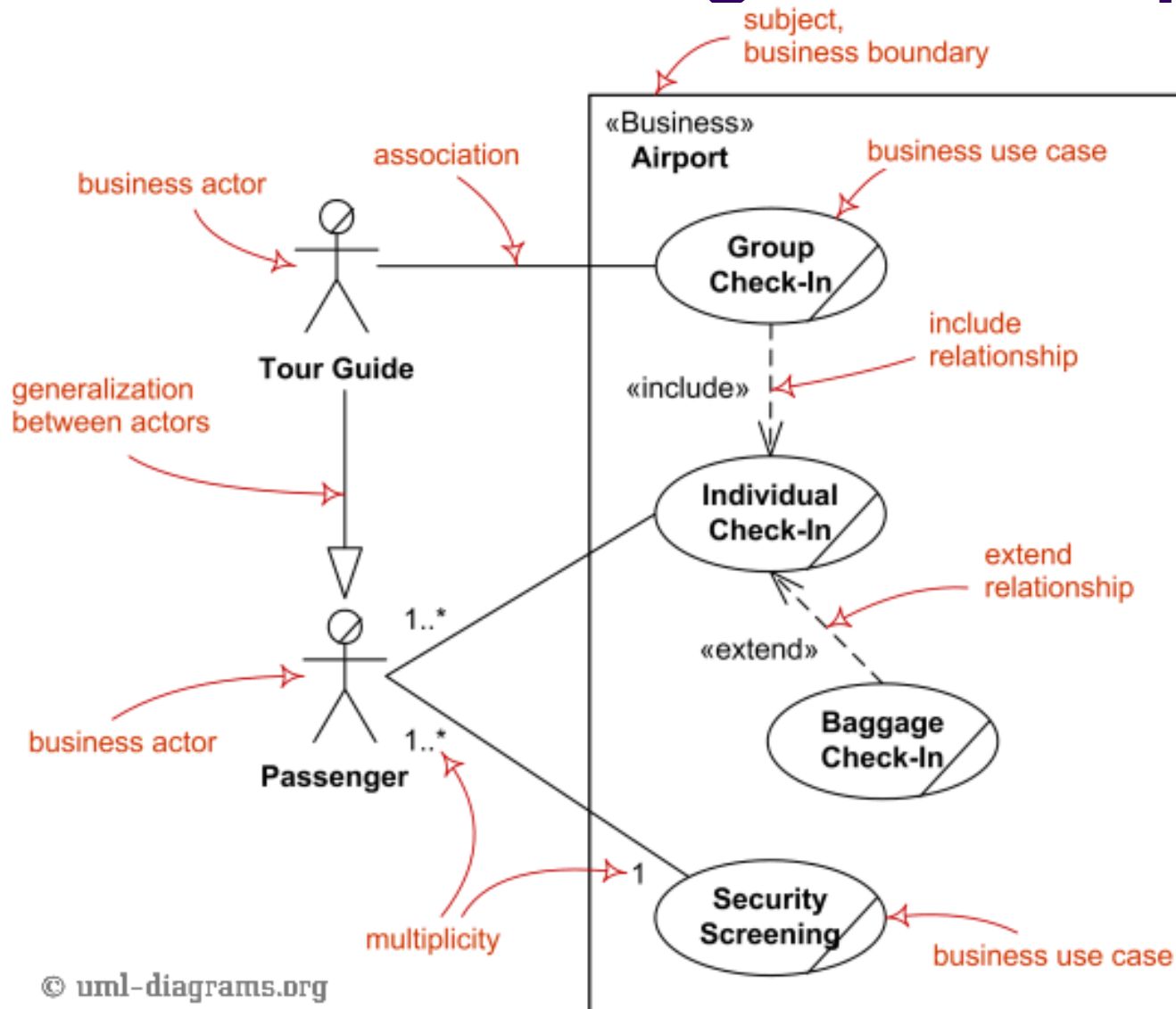


- While support for business modeling was declared as one of the goals of the UML, UML specification provides no notation specific to business needs.
- Business use cases were introduced in Rational Unified Process (RUP) to represent business function, process, or activity performed in the modeled business.
- A **business actor** represents a role played by some person or system external to the modeled business, and interacting with the business.
- A **business use case** should produce a result of observable value to a business actor.

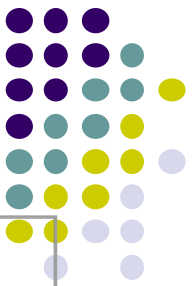
Source: <http://www.uml-diagrams.org/use-case-diagrams.html>



Business use case diagram example

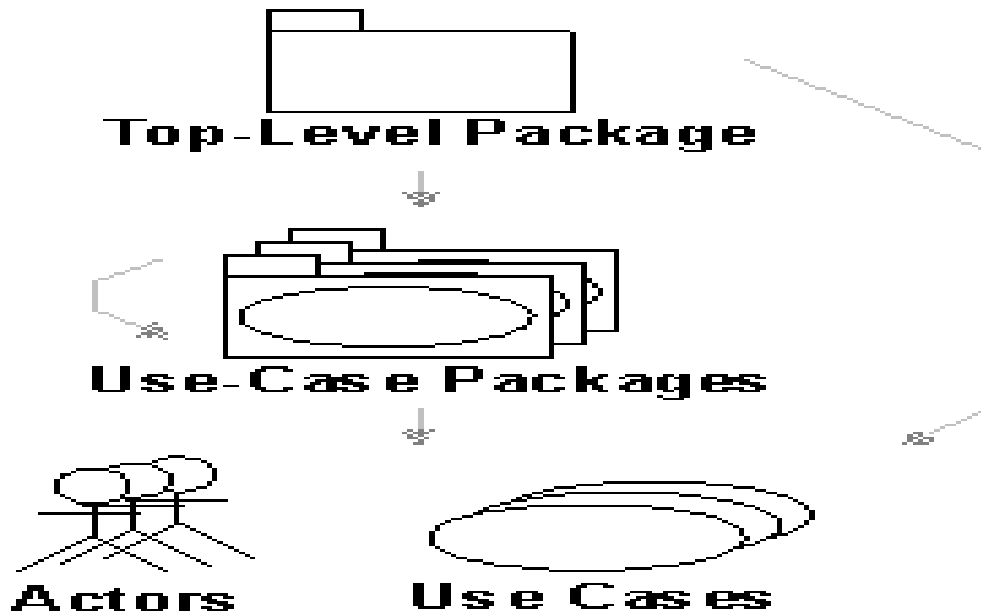


Use-Case Packages



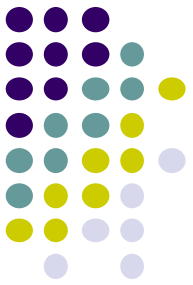
Use-Case Package

A **use-case package** is a collection of use cases, actors, relationships, diagrams, and other packages; it is used to structure the use-case model by dividing it into smaller parts.



A graph showing the use-case model hierarchy. Arrows indicate possible ownership.

Use cases and requirements capturing – concepts [Bruegge&Duttoit, 2004]

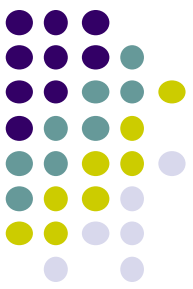


- Req. capturing is focused on the purpose of the system
- Req. spec serves as a contract b/w clients and developers
- Req. capture and analyses are concentrated on the user's view of the system
- Functional and non-functional req's (usability, reliability, adaptability, maintainability, performance, scalability...)
- Completeness, consistency, unambiguity and correctness – definitions?

Use cases and requirements capturing – concepts (2)



- Realism – the system req. lead to realization under constraints
- Traceability – each system function can be traced back to its corresponding set of req's
- Verifiability and validity – after construction phase, repeatable tests can be built for proving the system fulfils the req. spec. Examples for non-verifiable req's:
 - Our product shall have a ~~good~~ GUI.
 - The system should be ~~error free~~.
 - The response time will be less than 5 sec's for ~~most~~ of the cases providing there is assured ~~high bandwidth~~.
 -



Requirements capturing activities

- Identifying Actors
- Identifying Scenarios
- Identifying Use Cases
- Refining Use Cases
- Identifying Relationships among Actors and Use Cases
- Identifying Initial Analyses Objects
- Identifying Non-functional Req's

Identifying actors for FRIEND

[Bruegge&Dutoit, 2004]

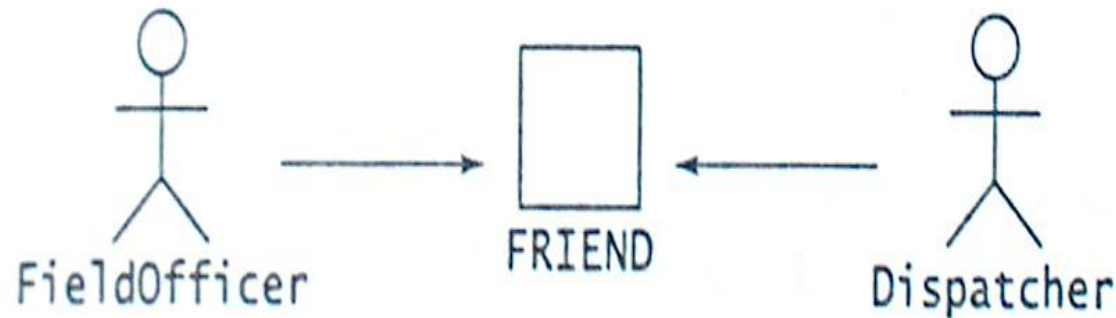
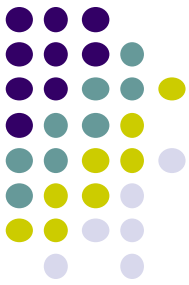


Figure 4-5 Actors of the FRIEND system. FieldOfficers not only have access to different functionality, they use different computers to access the system.

FRIEND = First Responder Interactive Emergency Navigational Database

Identifying use cases in FRIEND

[Bruegge&Dutoit, 2004]

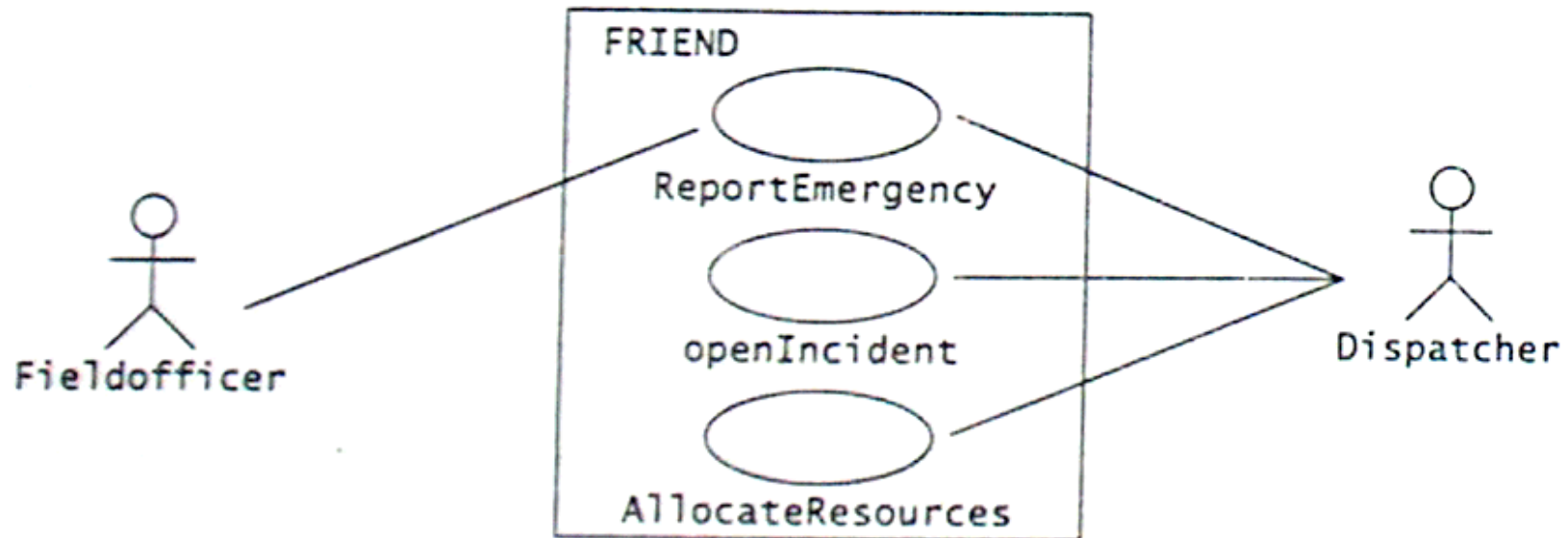
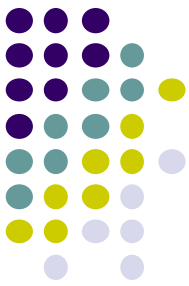


Figure 2-13 An example of a UML use case diagram for First Responder Interactive Emergency Navigational Database (FRIEND), an accident management system. Associations between actors and use cases denote information flows. These associations are bidirectional: they can represent the actor initiating a use case (FieldOfficer initiates ReportEmergency) or a use case providing information to an actor (ReportEmergency notifies Dispatcher). The box around the use cases represents the system boundary.

Refining use cases in FRIEND

- a bad and a good use case example

[Bruegge&Dutoit, 2004]



<i>Use case name</i>	Accident	<i>Bad name: What the user is trying to accomplish?</i>
<i>Initiating actor</i>	Initiated by FieldOfficer	
<i>Flow of events</i>	<ol style="list-style-type: none">1. The FieldOfficer reports the accident.2. An ambulance is dispatched.3. The Dispatcher is notified when the ambulance arrives on site.	<p><i>Causality: Which action caused the FieldOfficer to receive an acknowledgment?</i></p> <p><i>Passive voice: Who dispatches the ambulance?</i></p> <p><i>Incomplete transaction: What does the FieldOfficer do after the ambulance is dispatched?</i></p>

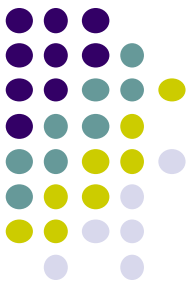
Figure 4-9 An example of a poor use case. Violations of the writing guide are indicated in *italics* in the right column.

FRIEND = First Responder Interactive Emergency Navigational Database

<i>Use case name</i>	ReportEmergency
<i>Participating actors</i>	Initiated by FieldOfficer Communicates with Dispatcher
<i>Flow of events</i>	<ol style="list-style-type: none"> 1. The FieldOfficer activates the "Report Emergency" function of her terminal. 2. FRIEND responds by presenting a form to the FieldOfficer. 3. The FieldOfficer fills out the form by selecting the emergency level, type, location, and brief description of the situation. The FieldOfficer also describes possible responses to the emergency situation. Once the form is completed, the FieldOfficer submits the form. 4. FRIEND receives the form and notifies the Dispatcher. 5. The Dispatcher reviews the submitted information and creates an Incident in the database by invoking the OpenIncident use case. The Dispatcher selects a response and acknowledges the report. 6. FRIEND displays the acknowledgment and the selected response to the FieldOfficer.
<i>Entry condition</i>	<ul style="list-style-type: none"> • The FieldOfficer is logged into FRIEND.
<i>Exit condition</i>	<ul style="list-style-type: none"> • The FieldOfficer has received an acknowledgment and the selected response from the Dispatcher, OR • The FieldOfficer has received an explanation indicating why the transaction could not be processed.
<i>Quality requirements</i>	<ul style="list-style-type: none"> • The FieldOfficer's report is acknowledged within 30 seconds. • The selected response arrives no later than 30 seconds after it is sent by the Dispatcher.

Figure 2-14 An example of a use case, ReportEmergency.

Documenting Flow of Events in Visual Paradigm



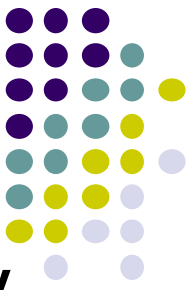
The screenshot shows the Visual Paradigm software interface with the 'Flow of Events' tab selected. The toolbar includes icons for creating new events, deleting, undo, redo, and other flow-related actions. The main area displays a list of events for a physical examination process.

- 1. Fill in physical examination form
 - 1.1. Download the physical examination form on the student visa website
 - 1.2. Double check filled information
 - 1.3. Make a cheque for medical fee
- 2. Call a physician to reserve an appointment
- 3. Arrive at the clinic at at the scheduled time
- 4. Follow the instruction of physician and X-ray physician
 - 4.1. Perform general physical examination
 - 4.2. Perform detailed physical examination
 - 4.3. Perform urine test
 - 4.4. Perform X-ray check

Extension:

- 3.a. For those who are late for physical examination
 - 1. Reschedule appointment with the physician
 - 2. Make an appointment with another physician

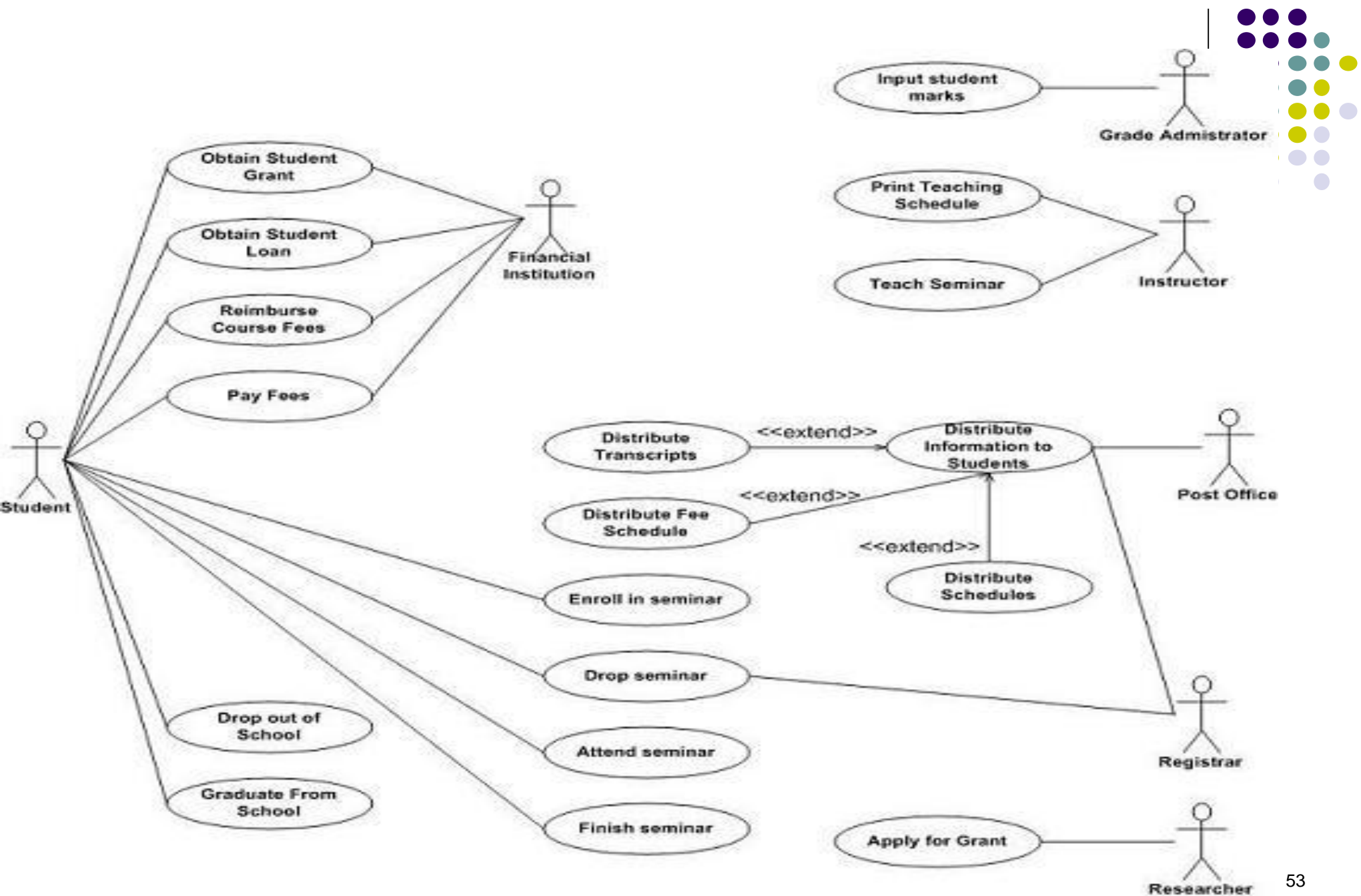
Example: The University Course Registration (UCR) Case Study

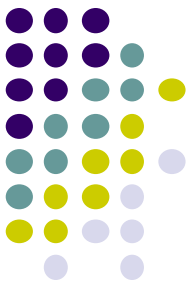


Goal: to assign automatically students to the courses teach by University professors.

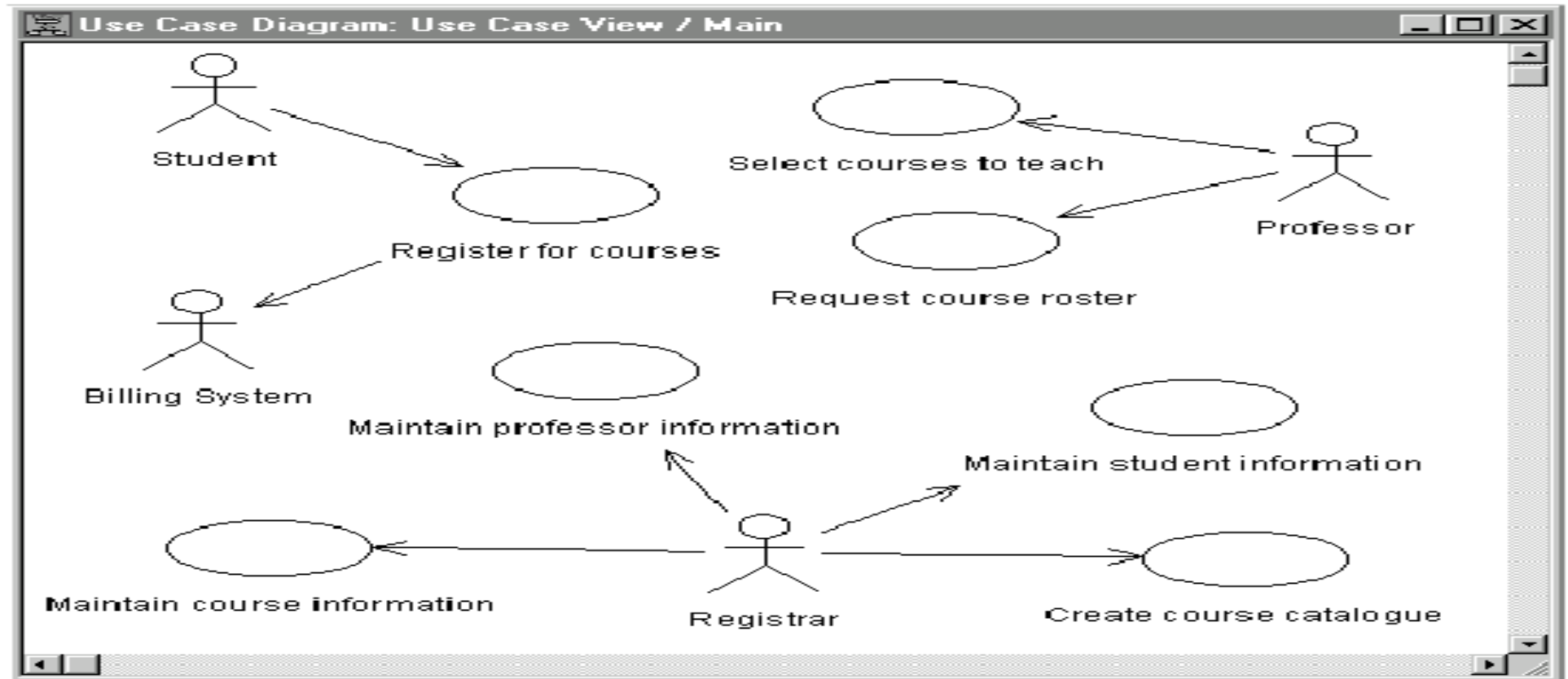
Current process:

- 1) Batch report printed with all the courses teach by the professors
- 2) Students filled out course registration form (no course offering for more than 20 students and less than five; four courses for each student)
- 3) Registrar's office processes the forms
- 4) Processing the conflicts – first choice usually is OK but in case of conflicts Registrar's officers talk to students to get additional choices
- 5) After successful assignment to courses, hard copy of the curriculum is sent to the students
- 6) Professors receive a student roster for each course they are scheduled to teach



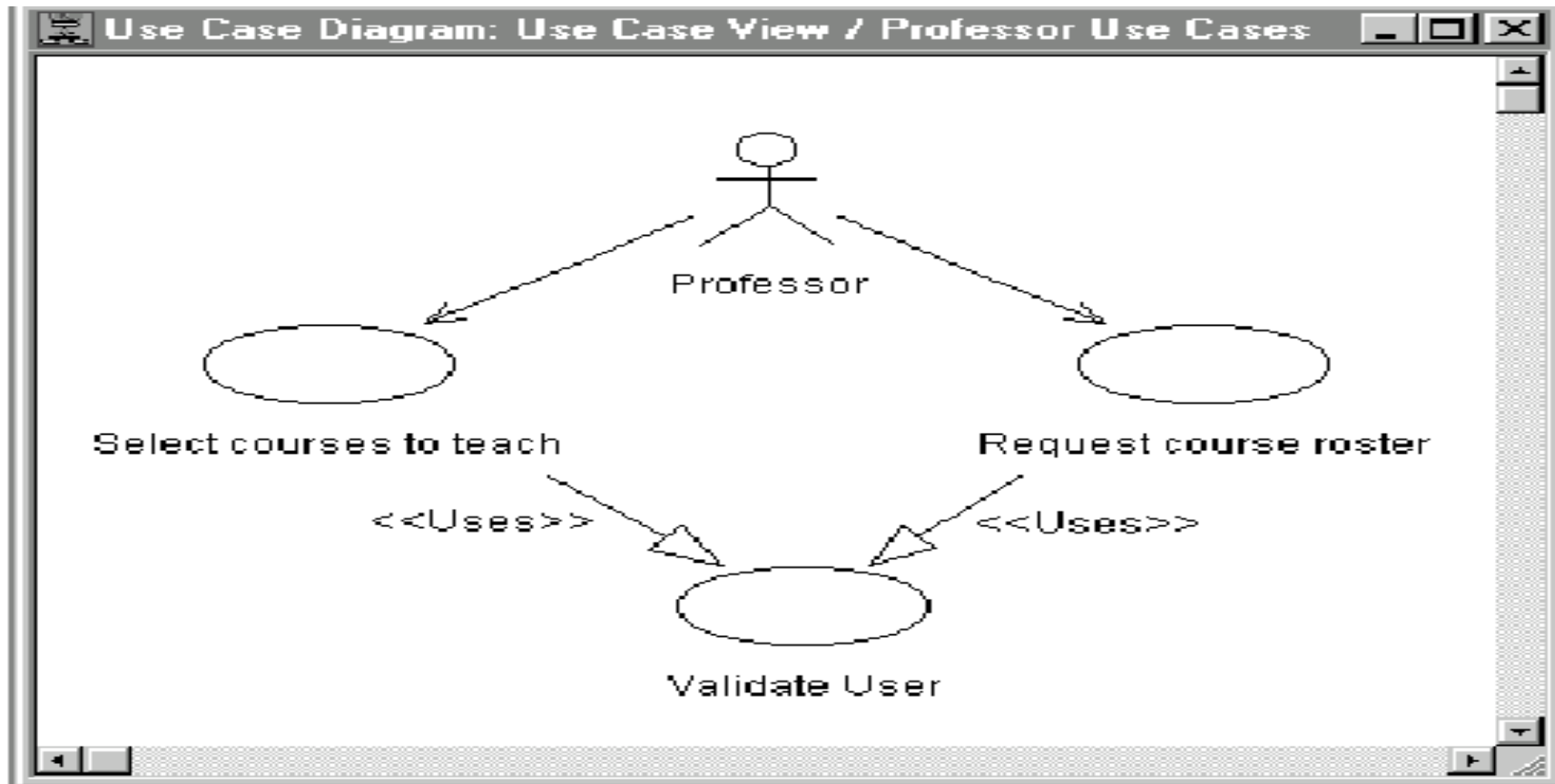
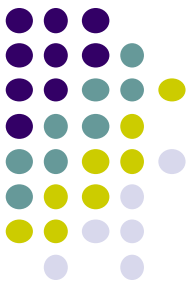


URC Use Cases



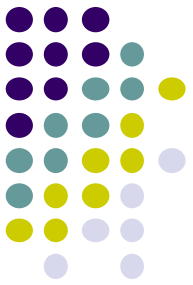
Main use case diagram

URC Use Cases - 2



An additional use case diagram

For Homework



Read the VP tutorials:

- Generation of use case scenario from notes- https://www.visual-paradigm.com/support/documents/vpuserguide/94/2575/83684_produceuseca.html
- Documenting use case details - https://www.visual-paradigm.com/support/documents/vpuserguide/94/2575/21179_documentingu.html