Metodologias de Desarrollo de Software I

Unified Modeling Language (UML)

Ivar Jacobson
Grady Booch
James Rumbaugh

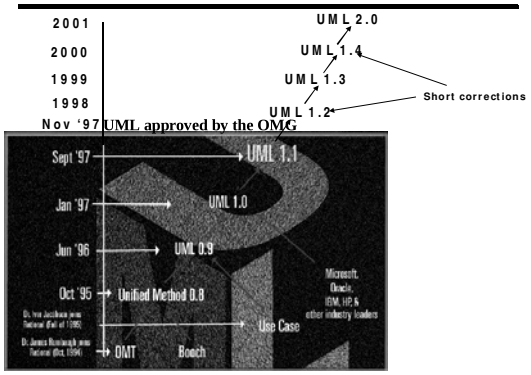
Agenda

- UML concepts
- UML evolution
- UML 1.4 vs UML 2.0
- Dynamic Part
  - UML 1.4 diagrams
  - UML 2.0 new concepts and diagrams
- Static Part
  - UML 1.4 diagrams
  - UML 2.0 new concepts and diagrams

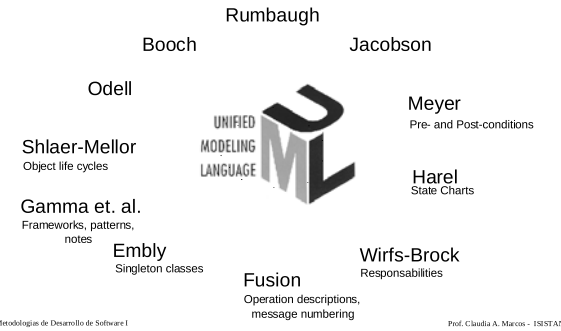
What is the UML?

- **UML stands for Unified Modeling Language**
- **The UML combines the best of the best from**
  - **Data Modeling concepts (Entity Relationship Diagrams)**
  - **Business Modeling (work flow)**
  - **Object Modeling**
  - **Component Modeling**
- **The UML is the standard language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system**
- **It can be used with all processes, throughout the development life cycle, and across different implementation technologies**

History of the UML



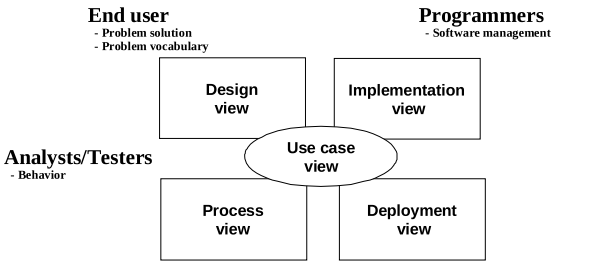
UML Supports Application Development



Architecture

A system architecture is perhaps the most important artifact to manage the iterative and incremental development of a system throughout its life cycle

Architecture



UML 1.4 Diagrams

Structural Diagrams

- Class diagram
- Object diagram
- Component diagram
- Deployment diagram

Behavioral Diagrams

- Use case diagram
- Sequence diagram
- Collaboration diagram
- Statechart diagram
- Activity diagram

UML 2.0 Diagrams

Structural Diagrams

- Class diagram
- Object diagram
- Component diagram
- Deployment diagram
- Composite Structure diagram
- Packages diagram

Behavioral Diagrams

- Use case diagram
- Sequence diagram
- Collaboration d. = Communication diagram
- Statechart diagram
- Activity diagram
- Interaction View diagram
- Timing diagram

UML Diagrams

- *Class diagram*: Models concepts in the application domain, as well as internal concepts invented as part of the implementation of the application
  - classes
  - relationships: association, generalization, dependency
- *Object diagram:* Models a set of objects and their relationships. Represents static snapshots of instances of the things found in class diagrams
- *Package diagram*: Models the classes packages

UML Diagrams

- *Deployment diagram*: Models the implementation structure of the application itself, such as organization into components and its deployment onto run-time nodes
- *Component diagram*: Models the organization of the model itself
- *Composite structure diagram*: Models a composite class and shows its internal structure

UML Diagrams

- *Use Case diagram*: Models the functionality of the system as perceived by outside users, called actors
  - actors
  - use cases
- *Interaction diagram*: Describes sequence of messages exchanges among roles that implement behavior of a system
  - sequence diagrams
  - collaboration (communication) diagrams

UML Models

- *Statechart diagram*: Models the possible life histories of an object of a class
  - States
  - Transitions
  - Events
  - Activities
- *Activity diagram*: Shows the computational activities involved in performing a calculation
- *Interaction view diagram*: models the activity and sequence diagrams together
- *Timing diagram*: models actions over time

Part I - Dynamic Part

- Use case diagram
- Activity diagram
- Sequence diagram
- Collaboration d. = Communication diagram
- Statechart diagram
- Interaction View diagram
- Timing diagram

Use Cases

- A use case specifies the behavior of a system or a part of a system and is a description of a set sequences of actions
- Use cases are used to capture the intended behavior of the system in develop
- A use case represents a functional requirement of a system as a whole

Use Cases - Concepts

- A use case involves the interaction of actors and the system
- An actor represents a coherent set of roles that use cases play when interacting with these use cases
- Typically, an actor represents a role that a human, a hardware device, or even another system plays with a system

Actor

Use case

Use Cases - Flow of Events

- A use case describes what a system does but it does not specify how it does it
- A use case is specified describing a flow of events in text clearly, including how and when the use case starts and ends, when the use case interacts with the actors and what objects are exchanged

Use Case - Basic Flow

Main flow of events. The use case starts when the system prompts the Customer for a PIN number. The Customer can now enter a PIN number via the keypad. The Customer commits the entry by pressing the Enter button. The system then checks this PIN number to see if it is valid. If the PIN number is valid, the system acknowledges the entry, thus ending the use case

Use Case - Alternative Flows

Exceptional flow of events. **If the Customer enters an invalid PIN number, the use case restarts. If this happens three times in a row, the system cancels the entire transaction, preventing the Customer from interacting with the ATM for 60 seconds**

Use Case - Alternative Flows

Exceptional flow of events. **The Customer can cancel a transaction at any time by pressing the Cancel button, thus restarting the use case. No changes are made to the Customer´s account.**

Exceptional flow of events. **The Customer can clear the PIN number anytime before committing it and reenter a new PIN number.**

Use Cases Documentation

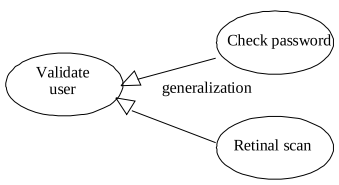
- **Textual description**
- **Enumerated description**
- **Pre, pos conditions**
- **Basic and alternative flow of control together / in different descriptions**
- **Use Cases extensions and include describe with the main use case**
- **.....**

Use Cases - Relationships

- ***Generalization*** **among use cases is just like generalization among classes**
- **An *include* relationship between use cases means that the use case explicitly incorporates the behavior of another use case at a location specified in the base. The included use case never stands alone**
- **The *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**

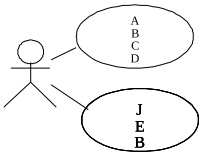
Use Cases - Generalization

- **As in a class diagram. If a use case can be developed in different ways so a generalization relationship is needed**
- **There are as use cases as different development use case alternatives are, and the father**



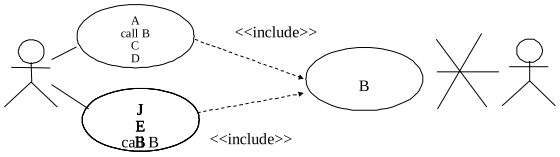
Use Cases - Include

- **When two or more use cases share a part of behavior so the shared behavior is factorized in a new use case**
- **The new use case is not a normal use case, that is it will not be activated by an actor (90%)**
- **When the original use case is activated the included use cases are activated too (always)**
- **The include relationship is "explicit"**



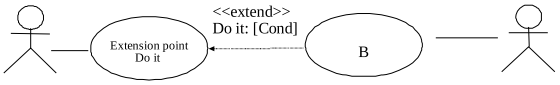
Use Cases - Include

- When two or more use cases share a part of behavior so the shared behavior is factorized in a new use case
- The new use case is not a normal use case, that is it will not be activated by an actor (90%)
- When the original use case is activated the included use cases are activated too (always)
- The include relationship is “explicit”



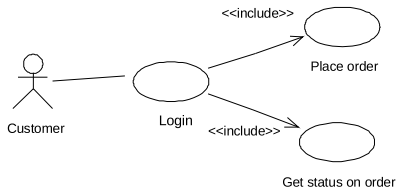
Use Cases - Extend

- If a condition is true a use case needs another to complete its behavior
- The new use case is a normal use case, that is it is activated by an actor (90%)
- When the original use case is activated according to the condition the extend use case is activated
- The extend relationship is “implicit”



Login Use Case

- Login use case that includes all of the other use cases that are secure
- In addition to updating the diagram, it is also necessary to write the text of the login use case. None of the other use cases change.



Login Use Case

1. The use case begins when the customer starts the application
2. The system prompts the customer to enter a username and password
3. The customer enter a username and password
4. The system verifies that this is a valid user
5. While the customer does not select exit, do the following steps in any order
6. The customer may choose to place an order (include Place Order)
7. The customer may choose to get the status on an order (include Get Status on Order)

end loop.

8. The use case ends.

Login Use Case

Benefits

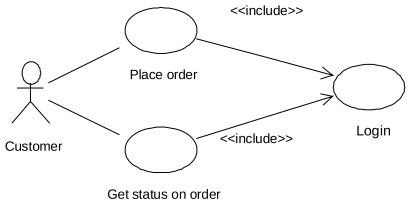
- The use case diagram looks like what the designer expect: The customer logs in, and form there he she can access any of the allowed system functions

Drawback

- The text of the use case is difficult to maintain in case a new use case is needed in the system. It is possible to forget to make the change to login
- It is not good that the login has the knowledge of all other parts of the system, Use cases have to be independent of each other

Login Use Case

- The other use cases include login
- It is necessary to update the use case diagram and the text



Login Use Case

Login Use Case

1. The use case begins when the customer starts the application
2. The system prompts the customer to enter a username and password
3. The customer enter a username and password
4. The system verifies that this is a valid user
5. The use case ends

Partial Place Order Use Case

1. The use case begins when the customer logs in to the system (include login)

Login Use Case

Benefits

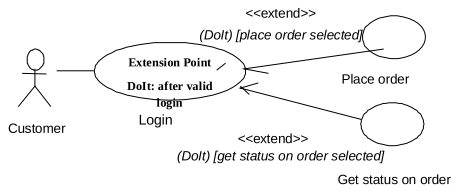
- The login use case describes login and nothing else

Drawbacks

- The Place Order use case and the diagram make it appear that the customer has to log in every time s/he wants to do something

Login Use Case

- The other use cases extend the login use case
- It is necessary to update the use case diagram and the text



Login Use Case

1. The use case begins when the customer starts the application
2. The system prompts the customer to enter a username and password
3. The customer enter a username and password
4. The system verifies that this is a valid user
5. While the customer does not select exit
6. Extension point: Do It
7. The use case ends

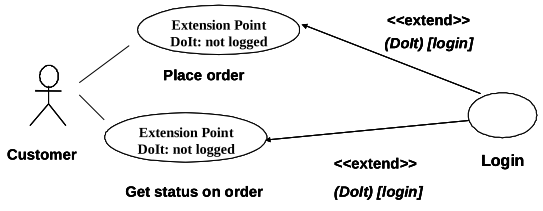
Login Use Case

Benefits

- The customer logs one time and from there gets access to the resto of the system. Login adds an extension point, but that is all, it is no necessary to change it when adding new use cases
- **Drawbacks**
- The description of the extentio. It can be a difficult relationship to explain, especially to peole who are not developers

Login Use Case

- The login extend the other use cases
- It is necessary to update the use case diagram and the text



Login Use Case

1. The use case begins when the customer starts the application
2. The system prompts the customer to enter a username and password
3. The customer enter a username and password
4. The system verifies that this is a valid user
5. While the customer does not select exit
6. The use case ends

Place Order Use Case

1. Extension point: Do It

Get Status on Order Use Case

1. Extension point: Do It

Login Use Case

Benefits

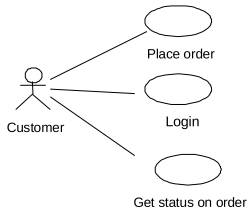
- The login only describes login and nothing else

Drawbacks

- The description of the extension relationship can be a difficult to explain, especially to peole who are not developers
- All the use cases has to adds the extension point

Login Use Case

- The login use case can be completely independent of the other use cases, but a precondition is included in the other use cases that a valid user be logged in before they can be executed
- It is necessary to update the use case diagram and the text



Login Use Case

1. The use case begins when the customer starts the application
2. The system prompts the customer to enter a username and password
3. The customer enter a username and password
4. The system verifies that this is a valid user
5. The use case ends

Partial Place Order Use Case

Assumption: A valid user has logged into the system

Login Use Case

Benefits

- The login only describes login and nothing else
- The diagram and the text are clear and easy to understand and the system is more flexible
- Place Order does not require login to execute, but just a valid user. Executing login is one way to get a valid user, but there may be other validation methods as well

Login Use Case

- The last approach seem to be the easiest to read and it has the most flexibility
- However, all the approaches are correct, so the developer can pick the one that works best for his/her project

Documenting Create, Read, Update, Delete

- This kind of use cases are often necessary in applications that include maintaining data
- Two approaches:
  - Create a separate use case for each kind of access to the data
  - Create one use case for a data type that embeds all of the CRUD functions
- When the use case is named by the behavior that the user expects, they are not called Create Order, Read Order, Update Order and Delete Order. The developer uses names that makes sense to the users of the system

Documenting Create, Read, Update, Delete

- It is possible to make one use case called something like Maintain Orders, which have been in charge of anything having to do with orders. This does not really make sense because different actors use separate CRUD functions, so it makes more sense to make them separate use cases
- On the other hand, in some applications combining them may be sensible, for example, if the application is for a database administrator to update tables in a database
- The best way is to begin with separate use cases and them analyze if the can be merged to make them easier to read and maintain

Use Cases Documentation (1)

- Identifier: Unique use case identifier
- Author: Worker who develop the use case description
- Description: Set of sentences describing the use case ideas
- Primary actor: Use case initiator
- Secondary actor: Participating actors
- Assumption: Conditions assumed to be true. These conditions are not verified by the use case.
- Pre-condition: Similar to supposition. However, the conditions are verified by the use case.
- Post-condition: Conditions to be true at the end of the use case

Use Cases Documentation (2)

- Frequency: How frequent is the use case developed (once a day, every week, 500 times a day)
- Basic flow: Normal execution process flow (ideal case)
- Alternative flow: Deviations of the normal thread of control
- Termination: situation in which the use case finishes
- Included use cases: List of use cases that this use case include
- Use cases that this extend: List of use cases that this use case extend
- Non-functional requirements: List of non-functional requirements

Use Cases - Scenarios

- One sequence diagram is used to specify a use case's main flow, and variations of that diagram to specify a use case's exceptional flow
- A scenario is a specific sequence of actions that illustrates behavior
- Scenarios are to use cases as instances are to classes, that is a scenario is basically one instance of a use case

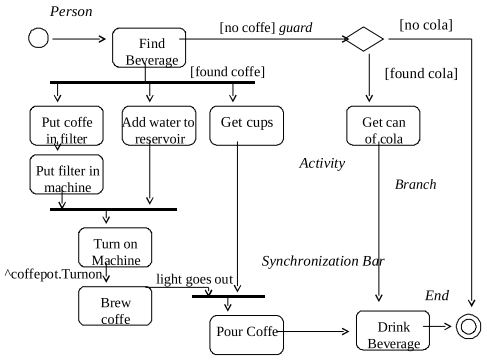
Activity Diagrams

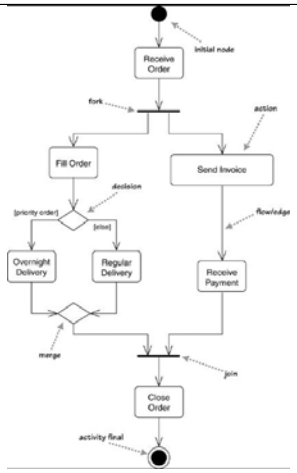
- An activity diagram is essentially a flowchart, showing flow of control from activity to activity
- An activity diagram looks at the operations that are passed among objects
- An activity diagram describes what happens but they do not tell how does what (swimlanes)

Activity Diagrams

- **Activity diagrams are used to analyze use cases, understand workflow across many use cases**
- **Activity diagrams are not used to try to see how objects collaborate, try to see how an object behaves over its lifetime**
- **Graphically, an activity diagram is a collection of vertices and arcs**

Activity Diagram

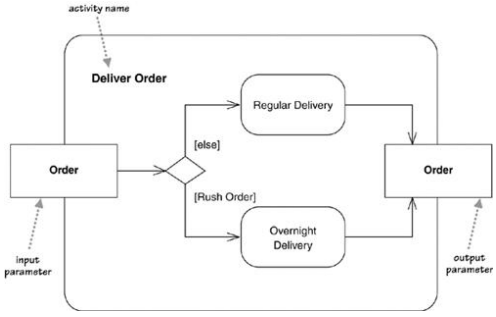


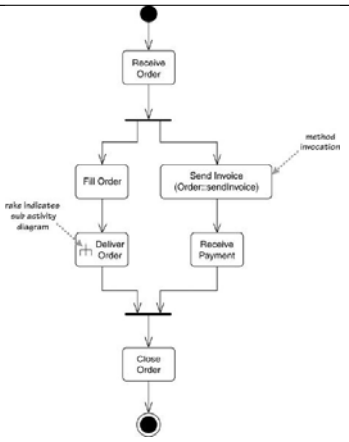


Activity Diagram 2.0

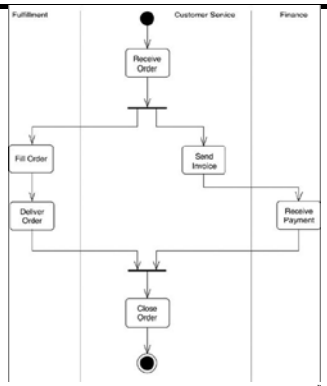
- Activity diagrams are a technique to describe procedural logic, business process, and work flow.
- It was considered similar to a statechart however it is not possible to model worflows so they are considered differente in UML 2.0
- Nodes on an activity diagram are called actions
- An activity refers to a sequence of actions, so the diagram shows an activity that's made up of actions
- Actions can be decomposed into subactivities showed in another diagram
- Actions can be implemented either as subactivities or as methods on classes
  - Subactivities are showed as rake symbol
  - Method with syntax class-name::method-name

Actions decomposed into subactivities





Swim lanes 1.4 / partitions 2.0



Interactions

- An interaction is a behavior that comprises a set of messages exchanged among a set of objects within a context to accomplish a purpose
- A message is a specification of a communication between objects
- Interactions are used to model the dynamic aspect of collaborations among objects

Interactions

- Well-structured interactions are like well-structured algorithms, efficient, simple, adaptable, and understandable
- An interaction statically sets the stage for its behavior by introducing all the objects that work together to carry out some action
- Interaction diagrams are used to model the flow of control within an operation, a class, a component, a use case, or the system as a whole

Interactions - Concepts

- A link is a semantic connection among objects (an instance of an association)
- A message is the specification of a communication among objects
- It is possible to model several kind of actions
  - Call: invokes an operation on an object
  - Return: returns a value to the caller
  - Send: sends a signal to an object
  - Create: creates an object
  - Destroy: destroys an object

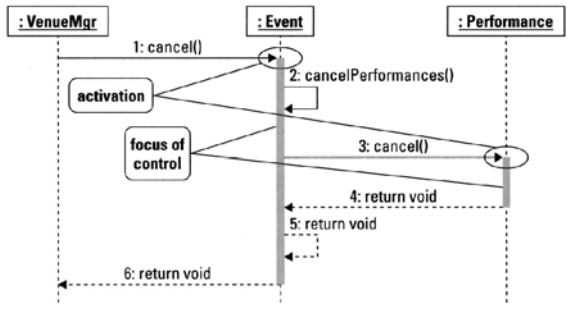
Interaction Diagrams

- An interaction diagram is used for modeling the dynamic aspects of systems
- An interaction diagram shows an interaction consisting of a set of objects and their relationships
- The interaction between objects can be visualized in two ways:
  - Emphasizing the time ordering of its messages (sequence diagram)
  - Emphasizing the structural organization of the objects that send and receive messages (collaboration diagram)

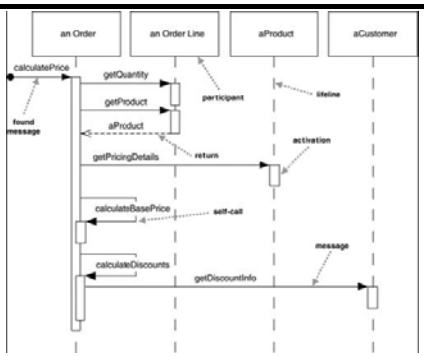
Interaction Diagrams

- A sequence diagram is an interaction diagram that emphasizes the time ordering of messages
- A collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages
- Interaction diagrams commonly contain:
  - Objects
  - Links
  - Messages

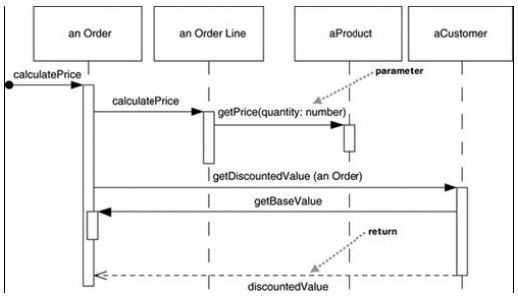
Object Activation



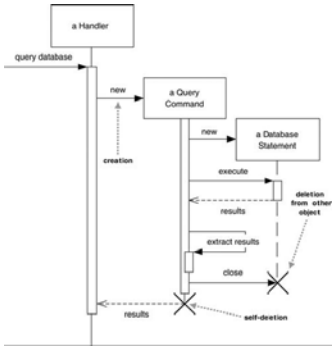
Sequence Diagram – Centralized Control



Sequence Diagram - Distributed Control



Sequence Diagram – Creating and Deleting



Sequence Diagrams UML 2.0

- Interaction frames: some region of the sequence diagram that is divided into one or more fragments.
- Each frame has an operator and each fragment may have a guard.
- For conditional logic the operator alt is used with the condition on each fragment. Only fragment whose guard is true will execute.

Sequence Diagrams

alt Alternative multiple fragments, only theone whose condition is true will execute

opt Optional, the fragment executes only if the supplied is true

par Parallel, each fragmen is run in parallel

loop Loop, the fragment may execute multiple times, and the guard indicates the basis of iteration

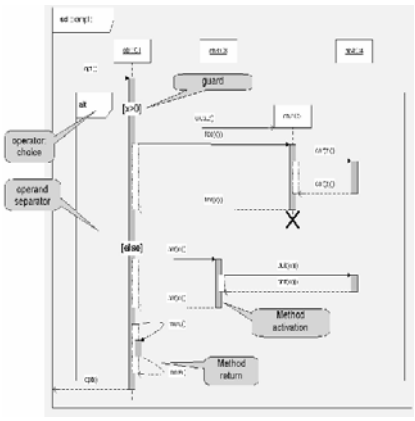
Sequence Diagrams

region Critical region, the fragmen can have only one thread executing it at once

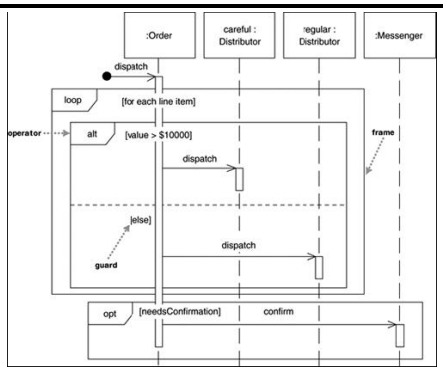
ne Negative, the fragment shows an invalid interaction

ref Reference, refers to an interaction defined on another diagram

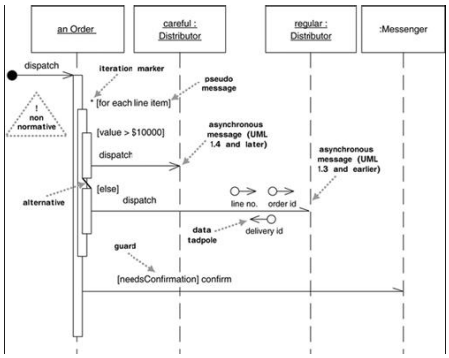
ad Sequence diagram, used to surround an entire sequence diagram, if you whish



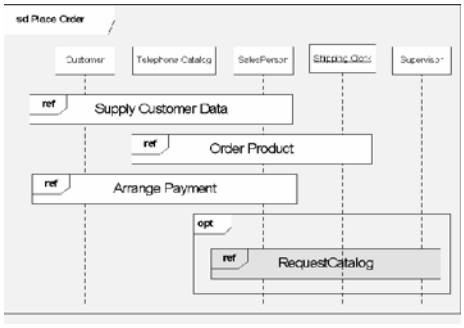
Loops and Conditionals



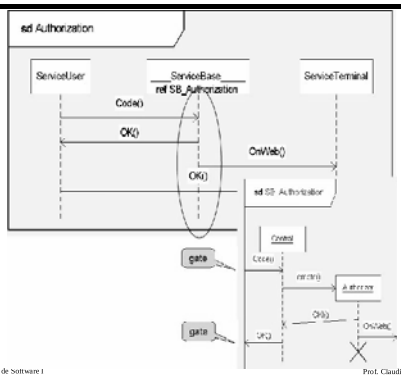
Loops and Conditionals UML 1.4



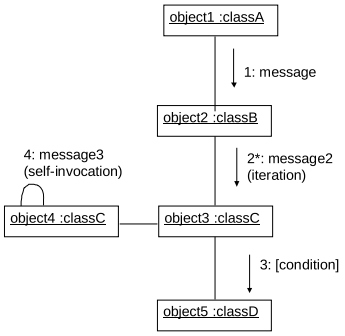
Ref Operator - Interactions Reuse



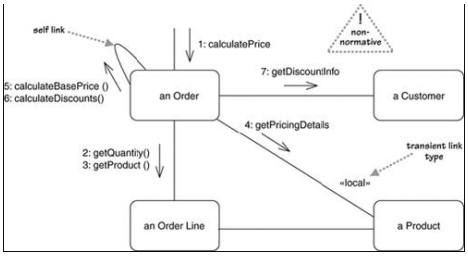
Decomposition of Lifelines

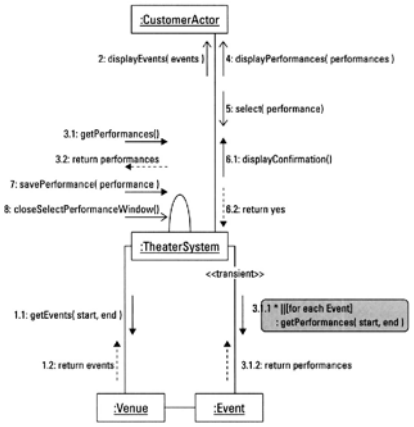


Collaboration Diagram 1.4 = Communication Diagram UML 2.0



Collaboration Diagram 1.4 = Communication Diagram UML 2.0





Parallel and Iterative behaviour

UML 2.0

Events

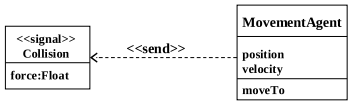
- **An event is the specification of a significant occurrence that has a location in time and space**
- **An event is an occurrence of a stimulus that can trigger a state transition**

Events

- **Events can be:**
  - **External, those that pass between the system and its actors**
  - **Internal, those that pass among the objects that live inside the system**
- **There are four kinds of events:**
  - **Signals**
  - **Calls**
  - **Passing of time**
  - **Change in state**

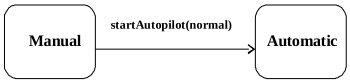
Signal Events

- **A signal represents a named object that is dispatched (thrown) asynchronously by one object and then received (caught) by another**
- **A signal may be sent as:**
  - **the action of a state transition**
  - **the sending of a message in an interaction**
  - **the execution of an operation**
- **A signal may have instances, relationships, attributes, and operations. Attributes serve as its parameters**



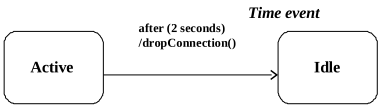
Call Events

- A call represents the occurence of a signal and the dispatch of an operation
- A call may trigger a state transition in a state machine
- A call is synchronous. When an object invokes an operation on another object that has a state machine, control passes from the sender to the receiver, the transition is triggered by the event, the operation is completed, the receiver transitions to a new state, and control is returns to the sender



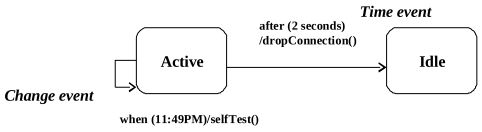
Passing of Time Events

- A time event is an event that represents the passage of time
- A time event is modeled by using the keyword *after* followed by some expression that evaluates to a period of time



Change Events

- A change event is an event that represents a change in state or the satisfaction of some condition
- A change event is modeled using the keyword *when* followed by some Boolean expression



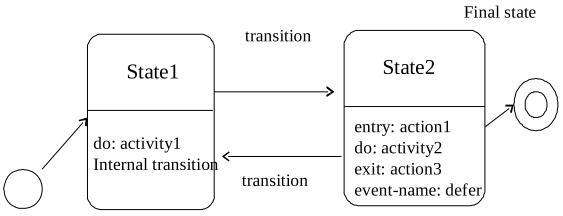
Statechart Diagram

- A statechart shows a state machine, emphasizing the flow of control from state to state
- A state machine is a behavior that specifies a sequence of state an object goes through during its lifetime in response to events, together with its responses to those events
- When an event occurs some activity will take place, depending on the current state of the object and changing its state
- Activity diagram shows flow of control from activity to activity, a statechart diagram shows flow of control from state to state

Statechart - States

- A state is a condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event
- A state has several parts:
  - *Name*
  - *Entry/exit actions*
  - *Internal transitions*
  - *Substates*
  - *Deferred events*

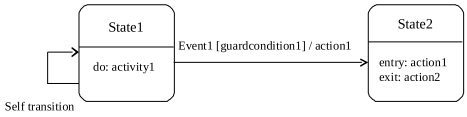
Statechart - States



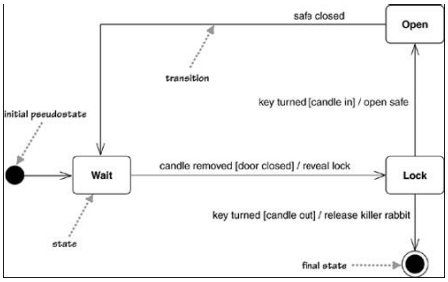
Initial state

Statechart - Transitions

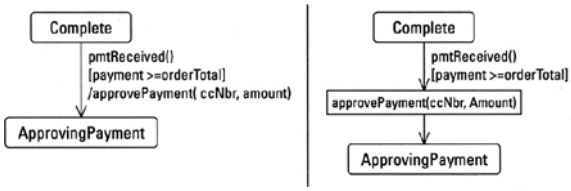
- **A transition is a relationship between two states indicating that an object in the first state will perform certain actions and enter the second state when a specified event occurs and specified conditions are satisfied**
- **An action is atomic, can not be interrupted by an event, in contrast to an activity which may be interrupted**



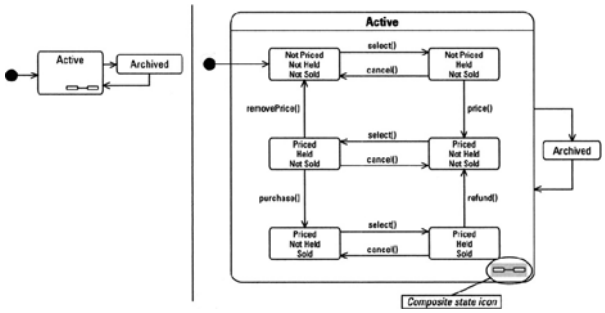
Statechart Diagram



Statechart Diagram UML 2.0



Composite States UML 2.0



New UML Diagrams

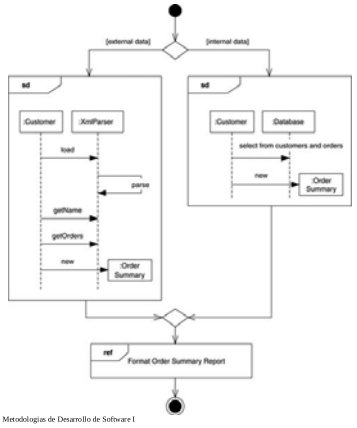
UML 2.0

Interaction View diagram

Timing diagram

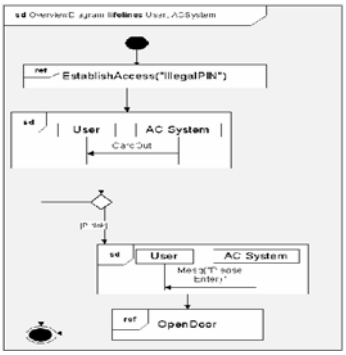
Interaction Overview Diagrams

- Interaction overview diagramas are a grafting together of activity and sequence diagramas
- The activities in a activity diagram are replaced by a little sequence diagrams
- Activity diagram serve as a “master chart” for a set of sequence diagrams
- This allow to construct a “map” of sequence diagrams and navigate easily among them
- Activity diagrams are based in UML 2.0 on Petri nets, rather than finite state automata



Interaction Overview Diagrams

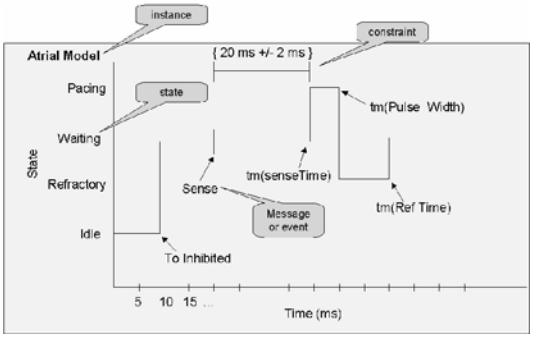
Interaction Overview Diagrams



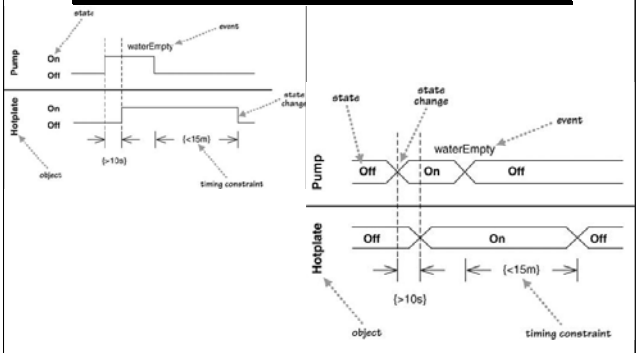
Timing Diagrams

- Timing diagram show actions over time
- Timing diagram has state long vertical axis and time along the horizontal
- Values along the vertical axis are normally discrete, such as “state” or the values of some numerate type
- The discrete value axis is quite general and any discrete evaluable condition may be used, even a logic or arithmetic expression
- Other elements can be show, such as gates, messages, and constratins

Timing Diagrams



Timing Diagrams



Timing Diagrams – Multiple Instances