UML: Behavioral viewpoint
Objectives and outcomes

• The objective of behavior modeling is to study/specify the behavior of the objects in a system.

• **Reminder**: An OO system is a system made up of objects that work together to realize some desirable functionality.
  - We have the desired functionality -&gt; **use cases**
  - We have the object structure -&gt; **classes**
Objectives and outcomes

• At the logical level, the behavior models allow us to:
  – Complete the structural model by finding the methods of our classes.
  – Validate the structural model by making sure that all required attributes and (navigable) associations are present.

• At the physical level,
  – Sequence & Activity diagrams define a specification for our algorithms.
  – State Machines can be used to generate executable code.
Objectives and outcomes

• We will study three complimentary behavior models:

  - **Sequence diagrams** specify how objects work together to realize a single functionality ➔ **Inter-Object view**

  - **Activity diagrams**: give a more workflow-oriented representation of how the work is likely to be done (actors, activities and dataflow) ➔ **Inter-Object view**

  - **State Machines** specify the global behavior (participation in all functionalities) of a single object ➔ **Intra-Object View**
Sequence Diagram
Sequence diagrams

- A sequence diagrams (also called “interaction diagrams”) shows a sequence of messages exchanged by the objects of a system.

- We generally use a sequence diagram to specify the realization of a single course of action in a use case.
  - Helps us find the methods of our classes
  - Helps us validate that the logical data structure is sufficient to realize the functionality.

- Very used in the industry => can be used to specify tests

- **Important**: a sequence diagram represents an instance of a possible interaction!!! It’s not an exhaustive representation of the system’s behavior!
Example

- **2 dimensions:**
  - Horizontal axe ⇒ objects
  - Vertical axe ⇒ time

![Diagram](image-url)
Sequence Diagram: Constituents

- **Objects**: Class’s instances that participate in the interaction

- Objects have lifelines

**Anonymous non typed object / Anonymous typed / non typed object / named and typed Object**
Sequence Diagram: Constituents

- **Messages**: a message is a communication between two objects to either exchange information or to trigger the execution of an action
  
  - Synchronous Message (a Call)
  
  - Asynchronous Message
  
  - Return Message
    
    - Optional (but a good practice to have it)
Creation & Destruction Messages
Other concepts
• **Activation**: thick box over object's life line
  - Either: that object is running its code or it is on the stack waiting for another object's method
Recursive calls

• An object can call its own operations
Control flow types

- With sequence diagrams one can easily identify (visually) the application’s control-flow type
  - Centralized Vs. Distributed
Fragments

- Used to represent loops, conditional branches, references to other sequences diagrams etc.

Example of a loop

Example of a conditional branch

Optional fragment
Fragments

- Example of the **ref** operator: used to make references between SD.
The previous UML notation for SD
Example: Chess Championship

• A simple application to manage an online chess championship
• Administrator
  – Create a championship
  – Generate games
• Participant
  – Register
• Player
  – Play a game
Example: Chess Championship

Chess Championship

- Administrator
  - Create Championship
- Participant
  - Generate Games
  - Register
- Player
  - Play
Example: Chess Championship

ChessChampionship
- close : boolean

operations
+ createChampionship()
+ register( partName : String )
+ generateGames()
+ sans nom1()

Player
- name : String

Game
- end : boolean

operations
+ makeMove( move : String )
- checkMat() : boolean
+ endGame()

With black Pieces

With white Pieces
Example: Chess Championship

• How to specify the behavior of this application using sequence diagrams?
• Idea:
  - At least, one sequence diagram per use case.
  - The sequence diagrams must be coherent with the class diagram.
Create a Championship

: Administrator

1: 

: ChessChampionship

2: 
Register two participants

1: register(partName="Xavier")

5: register(partName="-Fred")

6: 

7: 

8: 
Generate Games

1: generateGames()

2:

3:

4:
Play a game

1: makeMove(move=)

2: checkMat()

3: 

4: makeMove(move=)

5: checkMat()

6: 

7: endGame()
Sequence Diagrams: Conclusion

- Allow to describe the behavioral aspects of a system
- Used to formalize scenarios (from use cases)
- More focused on the Time aspect (messages order)
- Informs about the kind of the application’s control-flow visually
- Represents the link between the use cases (scenario) and class diagrams
  - Used to identify operations in the class diagram.
Collaboration Diagram
Collaboration Diagrams

- Same constituents as Sequence diagrams

- Focus: more on the objects involved in the interaction rather than on the temporal aspects (messages order)

- Not really used in projects

- Can be generated automatically from sequence diagrams (some tools)
Example
State Machines
State Machines

• **Attached to a class (object)**
  - More exhaustive than sequence diagrams
  - Describe how the object reacts to changes in its environment

• **Notions of states & transitions**
  - Inspired from David Harel work

• **Probably the more formal diagram in UML**
  - Very used in critical projects (aerospace, automotive, nuclear plants, etc.)
Example

Two main concepts:
- State
- Transition

<table>
<thead>
<tr>
<th>Door</th>
</tr>
</thead>
<tbody>
<tr>
<td>open()</td>
</tr>
<tr>
<td>close()</td>
</tr>
<tr>
<td>lock()</td>
</tr>
<tr>
<td>unlock()</td>
</tr>
</tbody>
</table>
A state is a "stage of existence" during which an object satisfies certain conditions, executes an activity, or waits for an event. The state defines how the object reacts to new events in its «life».

Examples:

<table>
<thead>
<tr>
<th>Class</th>
<th>Possible states</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human « age »</td>
<td>child, teenager, adult</td>
</tr>
<tr>
<td>Washing machine « cycle »</td>
<td>pre-wash, wash, spin, dry</td>
</tr>
<tr>
<td>FTP Server « connexion state »</td>
<td>wait for connexion, send data, wait for acknowledgement, resend data ..</td>
</tr>
<tr>
<td>Inhabitant “marital state”</td>
<td>single, married, divorced, widowed</td>
</tr>
</tbody>
</table>
State

• A state is described by a rounded rectangle containing:
  - **Name**, unique within the class;
  - **Entry** and **Exit** actions, instantaneous processes executed by the object on entering/exiting the state;
  - An **activity**, long-lasting processes executed while the object is in that state.
    • Often has the same name as the state, in which case it can be omitted.
Every state diagram has exactly one initial state, and any number of final states.

The **initial state** is the "point of creation" of the object.
- The transition leaving it is taken when the object is instantiated.
- This transition may contain a guard condition and an action, but not an event.
- It may have multiple mutually-exclusive outgoing transitions, but no incoming transitions.

The **final state** signifies that the object has terminated its execution, and has no further reason to exist.
- It has no outgoing transitions.
Transition

- A transition describes how an object moves from one state to another.
- Syntax:
  
  \[
  \text{event [ guard condition ] / action}
  \]

- The transition is taken \textit{when} the \textbf{event} is received \textit{if} the \textbf{guard condition} is true.
- When it is taken, the \textbf{action} is executed by the object.
Transition

- A transition without an event or guard condition is considered to be "automatic". It is taken when the state finishes executing its activity process.

- An event corresponds to a message received from the environment or another object during an interaction. It may contain data parameters which are received by the object.

- The guard condition is a boolean expression based on the object's internal attributes and the event's parameters.

- The action is an instantaneous process executed by the object when the transition is taken.
Examples of Events

• A condition becomes true
  – A guard condition

• Reception of signal from another object

• Reception of Call operation event

• Time condition
Examples of Events

• **EventName(Parameters)**

• **Time:** after(2 secondes)

• **Condition becomes true:** when(var = 3)
Internal Transitions

• The object remains in the same state

<table>
<thead>
<tr>
<th>Etat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ev1 [t=0]/ faire quelque chose</td>
</tr>
<tr>
<td>Ev1 [t&gt;0]/ faire autre chose</td>
</tr>
</tbody>
</table>

• The internal transition is triggered every time the event specified in the state is triggered

• This is different from a reflexive transition where the object goes out from the state then comes back
Guards : example

idle

when (temperature > 22) [winter]

when (temperature > 22) | [summer]

Fan

AC
Advanced Concepts
Composite states

- A state can be composed of other states
  - To ease the understandability of the problem
  - To factorize transitions
Example
Composite states

- Notation to hide the included states
Notion of concurrent states
Notion of concurrent states

- Another example
Super-state and history

- The **history** construct memorizes the last active sub-state of a super-state.
- It enables the system to leave the super-state and later return to the same sub-state.
- Classic example:

```plaintext
Idle ➔ Pre-wash ➔ Wash ➔ Spin ➔ Actif ➔ Door opened
```

- start
- open door
- close door
Composite Transition : Example

The diagram illustrates a state transition diagram with states labeled State0 and State1. There are transitions labeled e2[b < 0] from State0 to State2 and e1[b < 0] from State1 to State3. The diagram also shows transitions labeled [a < 0] from State0 to State3 and [a > 7] from State1 to State4. Additionally, there is a transition labeled [a = 5] from State3 to State4.
Activity Diagrams
Activity Diagrams

• Used to model Workflow or Business Processes

• Very expressive to model Use Cases
  - Instead of sequence diagrams for instance
    • More exhaustive and precise

• From UML 2.0, possible to model complex algorithms and operation’s bodies

• Can be attached to:
  – A class,
  – An operation,
  – A use-case (workflow)
Concepts
Activity Diagrams

• Since UML 2.0, we use the notion of action to designate atomic steps in a UML activity diagram (called activities in previous versions).

• An activity can be composed of:
  - Actions
  - Control Nodes (Fork, Merge, Decision, Join),
  - Object Nodes (input & output pins, object nodes, ActivityParameterNode)
  - Arrows (Control Flow & Object Flow)

• The execution model of UML AD is based on the notion of production/consumption of token
  - Largely inspired from Petri Nets

• Transitions are automatic
Notion of Activity Parameter Node

- Equivalent to the notion of Operation’s parameters
Notion of Call Actions

• An activity can be called from another activity (CallBehaviorAction)

• An activity can call a classical operation (CallOperationAction)
Signals

• Can be used to express a timeout for instance

• To send or to receive an event
  - To be caught by an AcceptEventAction
Swim Lanes

- Used to define the main roles of the business process (the WHO ?)
AD: Conclusions

• A very expressive diagram

• The specification is very complex. More than 350 pages in the standard

• In UML.2.0, the aim was to use AD as a programming language
  – Unfortunately, too complexes, many ambiguities
  – Only few tools are 100% compatible with the standard
  – Many semantics variation points
  – 10 lines of code remains more understandable than a huge AD
Lectures

- Software Engineering,
- The Mythical Man-Month
  - Frederick P. Brooks JR., Addison-Wesley, 1995
- Cours de Software Engineering du Prof. Bertrand Meyer à cette @:
  - http://se.ethz.ch/teaching/ss2007/252-0204-00/lecture.html
- Cours d’Antoine Beugnard à cette @:
  - http://public.enst-bretagne.fr/~beugnard/

-----------------------
- UML Distilled 3rd édition, a brief guide to the standard object modeling language
- UML2 pour les développeurs, cours avec exercices et corrigés
- UML 2 par la pratique, études de cas et exercices corrigés,
  - Pascal Roques, 6ème édition, Edition Eyrolles, 2008
- Cours très intéressant du Prof. Jean-Marc Jézéquel à cette @:
- La page de l’OMG dédiée à UML: http://www.uml.org/

------------------------
- Design patterns. Catalogue des modèles de conception réutilisables
  - Richard Helm (Auteur), Ralph Johnson (Auteur), John Vlissides (Auteur), Eric Gamma (Auteur), Vuibert informatique (5 juillet 1999), ISBN-10: 2711786447