Métodos de Desenvolvimento de Software (Software Development Methods) (MDS) 2016/2017

Interaction Diagrams

Motivation

Let’s continue the analysis process…

Previously…

- We have been studying:
  - Use Cases
  - Activity Diagrams
  - Domain Class Diagrams
  - OCL, to strengthen the business rules (including diagram integrity restrictions)
- We are now revisiting and analyze the use cases in greater detail (refine)
  - **Realizing Use Cases**

Goals for Use Case Realization

- To understand which analysis classes interact with each other in order to realize the behavior described in the Use Case
- To understand which message instances of those classes have to be exchanged in between classes in order to produce a certain behavior
  - Which are the main **operations** of the analysis Classes?
  - Which are the main **attributes** of the analysis classes?
  - Which are the main **relations** between analysis classes?
- If necessary, update the Use Case model, requirements, and analysis class model
  - **Essential to keep consistency !**

Use Case Realization

- Set of Classes that interact in order to realize the Use Case Behavior
  - e.g. In the “Borrow Book” Use Case, the librarian Use Case can interact with the “Book”, “Lending Registration” and “User” analysis classes to be able to realize the Use Case

Functional Requiriments Specification	High Level System Specification
Use Case	Analysis Class Diagram
	Interaction Diagram
	Special Requirements
	Refined Use Case

Use Case Realization

We are using the iterative method

- Analysis Class Diagrams tell a “story” about how the classes interact so that their instances work together in building the Use Case behavior
- The Interaction Diagrams show how class instances cooperate to realize that behavior
- We can find new requirements in the process of refining them
- The Use Cases can be refined even further

Interaction Diagrams

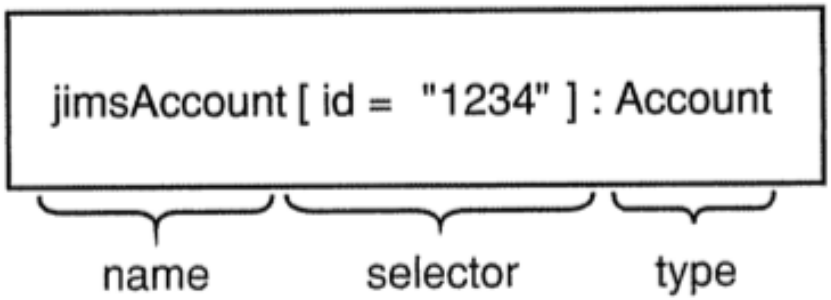
(Previously called Sequence Diagrams in UML 1.0)

Interaction

- Interaction are units of behavior of a classifier
  - The classifier sets the context to an interaction
  - In the Use Case realization, it is the Use Case the classifier that sets the context
  - When we detail an interaction, it is common to find new operations and attributes in the analysis classes
    - Therefore, we have to update and synchronize the class diagrams!

Main Elements in an Interaction

- Lifeline – a single participant in an interaction



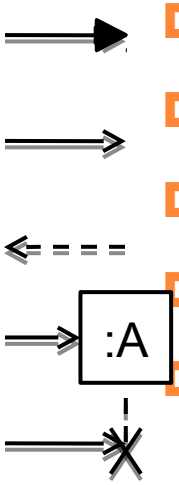
  - Name – used to refer to the lifeline
  - Type – Classifier name that the lifeline represents
  - Selector – Boolean expression to specify a particular participant instance (if it does not exist, the participant can be any instance of the corresponding type)
- Represents how a classifier instance acts in the interaction

Main Elements in an interaction

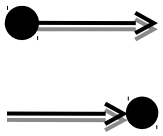
- Messages – represent the communication in between two lifelines in an interaction
  - Sent messages
  - Create and Destroy instances
  - Send signal

Messages

- Messages can be
  - Synchronous (who sends waits for answer)
  - Asynchronous (who sends does not wait for answer)
  - Return (returns focus of control)
  - Create (new lifeline)
  - Destroy (lifeline)



  - Found (unknown origin)
  - Lost (unknown destiny)



Interaction Diagrams

- Show communication in between objects
- Goal:
  - **To specify the Use Case realization**
  - Specify how to realize an operation
- Two types:
  - Sequence Diagrams
  - Collaboration Diagrams

Interaction (UML1.0 Sequence Diagrams) and Communication Diagrams (UML1.0 Collaboration Diagrams)

- Both specify the same information
- Each focuses in different aspect
  - **Sequence Diagrams (time oriented)**
    - Shows how messages are organized
    - Do not show how to get the reception object
  - **Collaboration Diagram (space oriented)**
    - Show the static and dynamic relations in between objects
    - The sequence of messages is explicitly shown
    - Time is not one dimension

Interaction Diagrams

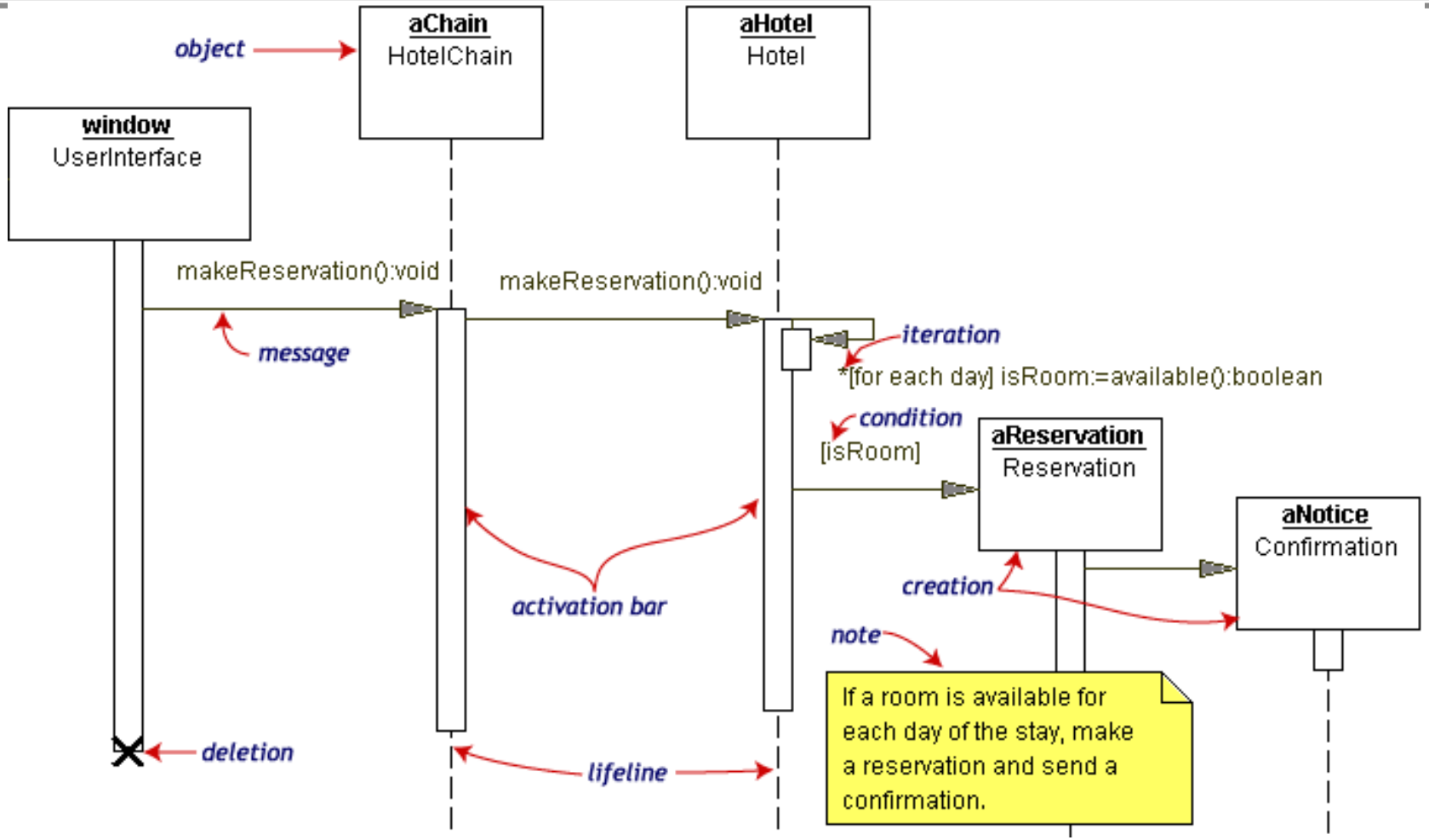
Interaction Diagrams

- Show the required communication (message exchange) between objects to execute a Use Case
- Source: Available information in the Use Case models or other (e.g. Activity Diagrams).
- As the Use case describes all the perspectives over a given functionality (including errors and exceptions), we can opt to build a Sequence Diagram per scenario.

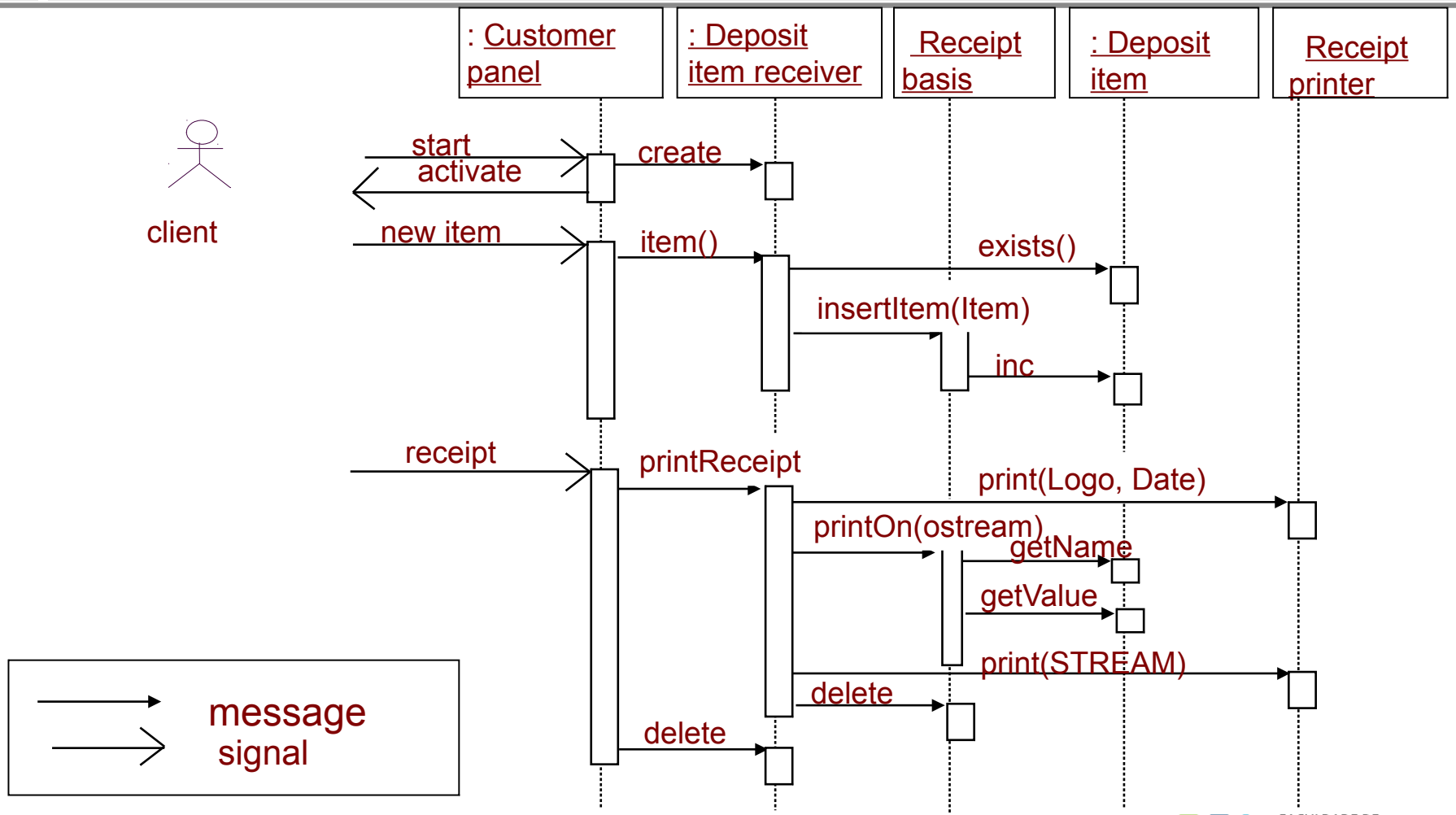
Interaction Diagrams: notation

- Objects (rectangles) organized along the X axis
- Object Lifeline: dashed line to represent the existence of an object in a given time length
- Messages (arrows), organized in time along the Y axis
- Control Flow: narrow rectangle to show the time length since the object receives the message until it answers (execution time of an operation);
  - This time can include execution times of sub-operations.

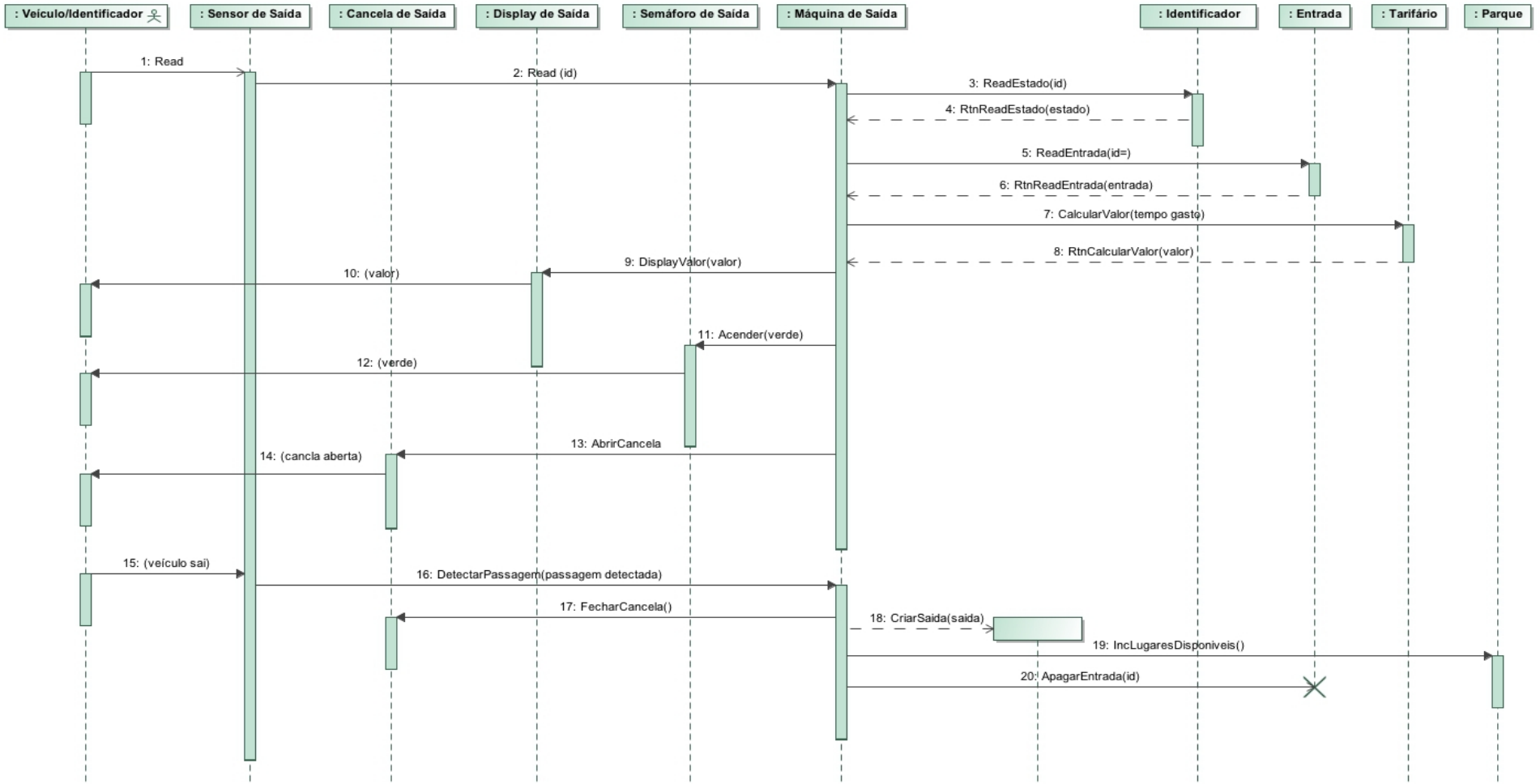
Interaction Diagrams: room reservation



Example: recycling machine



scenario “authorized car to leave the parking garage”

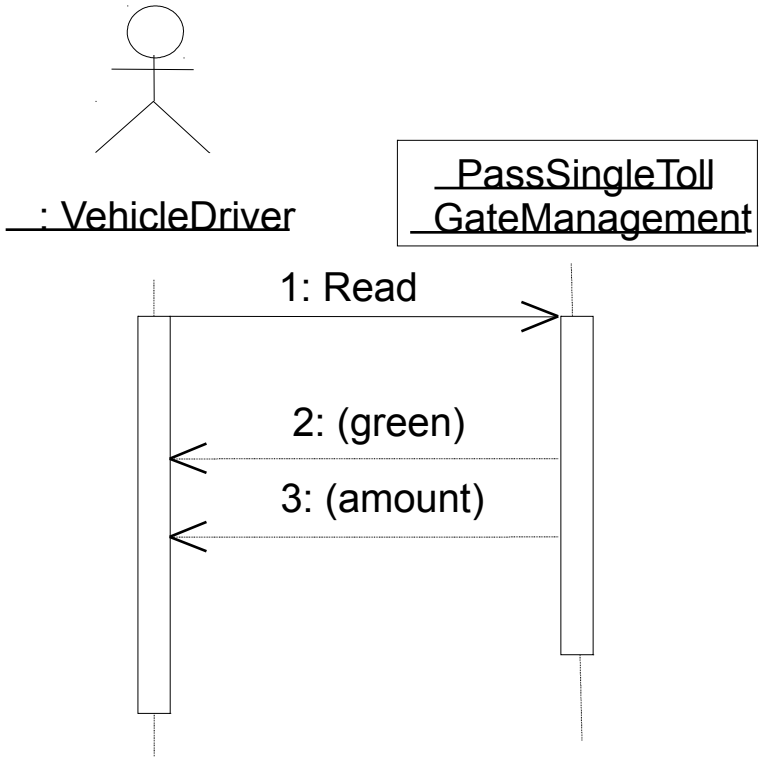


Problems while constructing

- It is not always obvious how to build the SD
  - Control objects are involved when we detect that the interface object leads/controls the execution;
  - The SD can start to be built by using the three types of objects as guides.
- Some construction rules:
  - The first message is always sent by an actor
  - The first message is always received by an interface object
  - Add a control object when the interface object becomes a decision maker

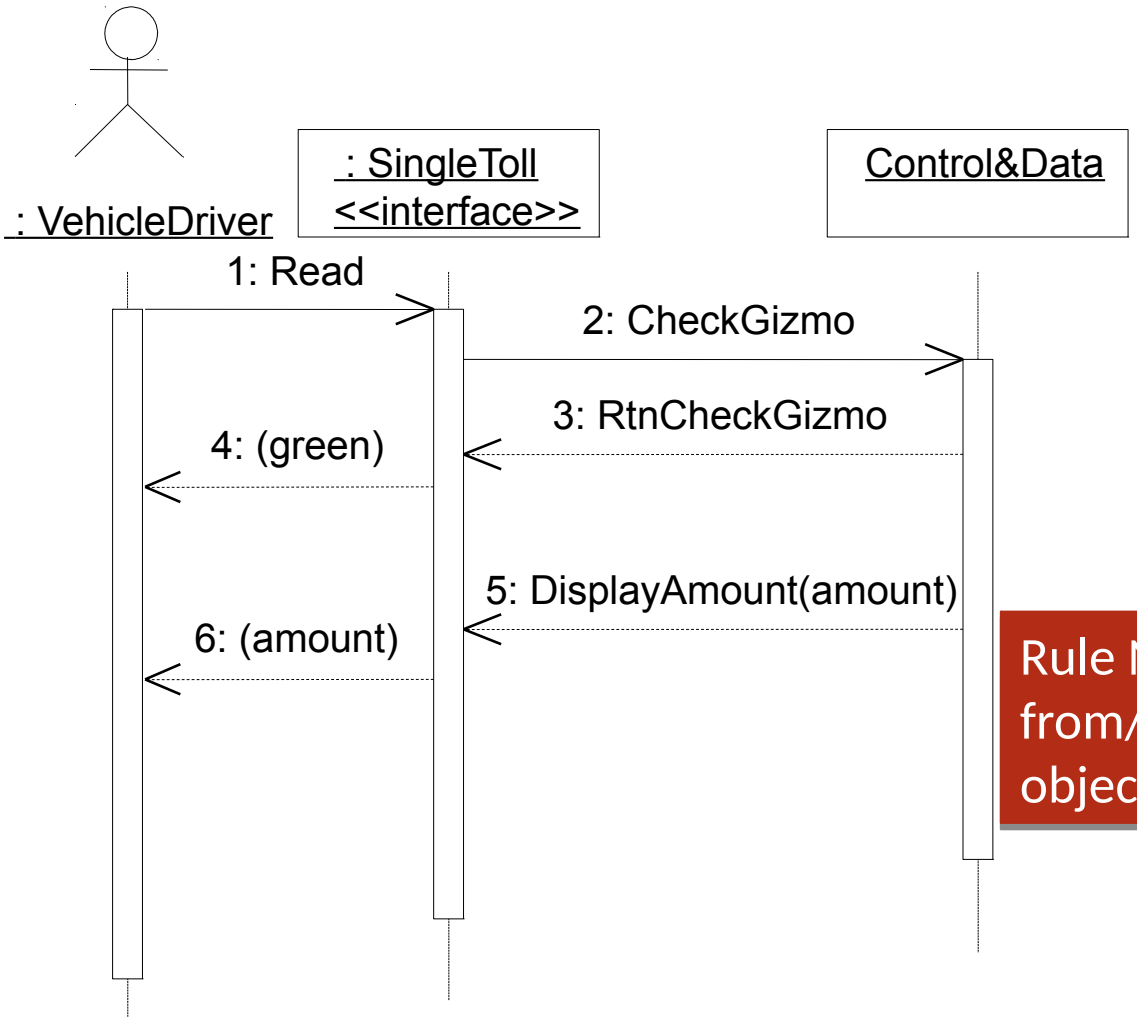
Step-by-step

Example: the car passes by the toll (e.g., “25 Abril” Bridge).



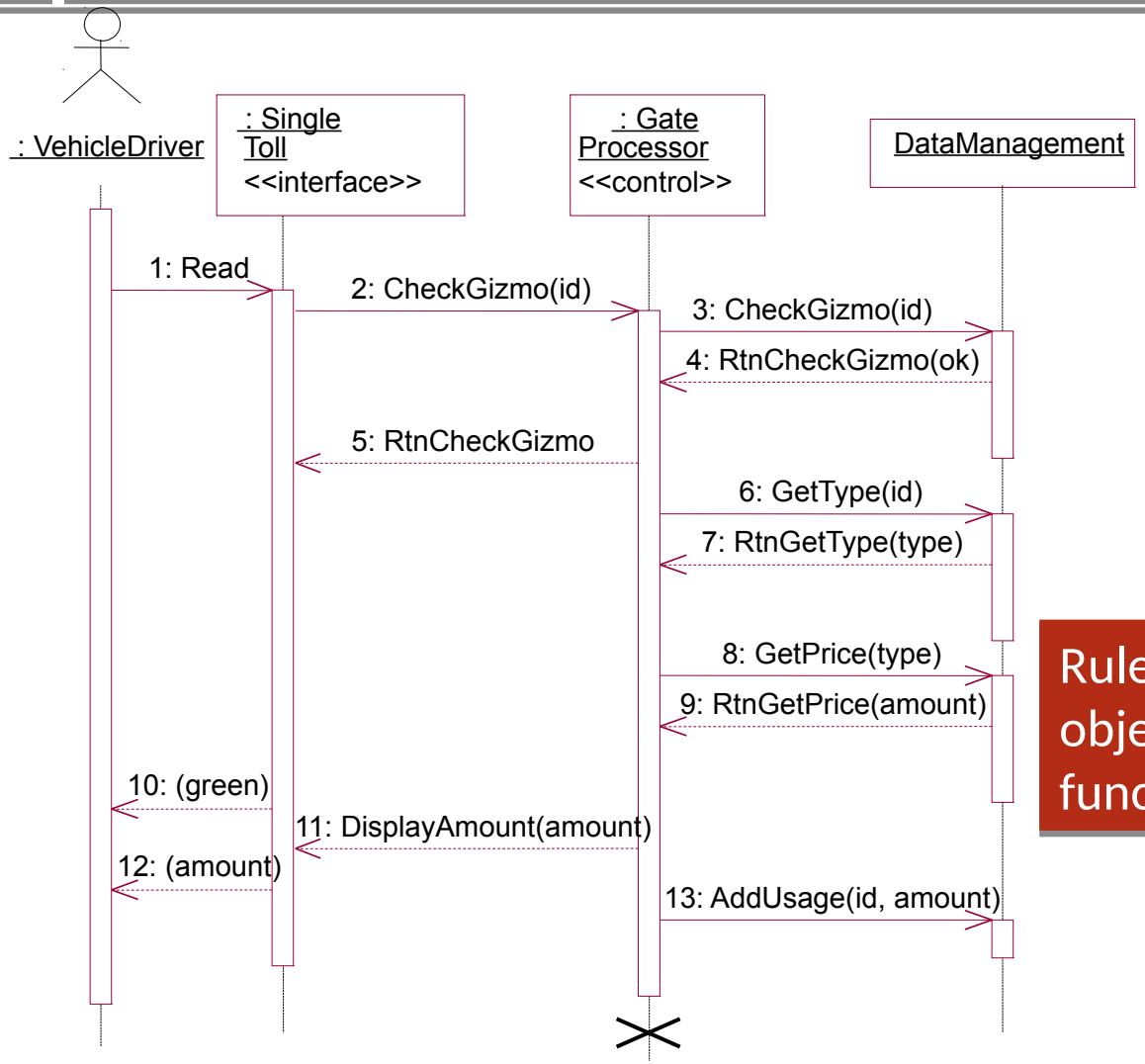
Rule Number 1: when starting, the system is seen as a black box.

with an interface object



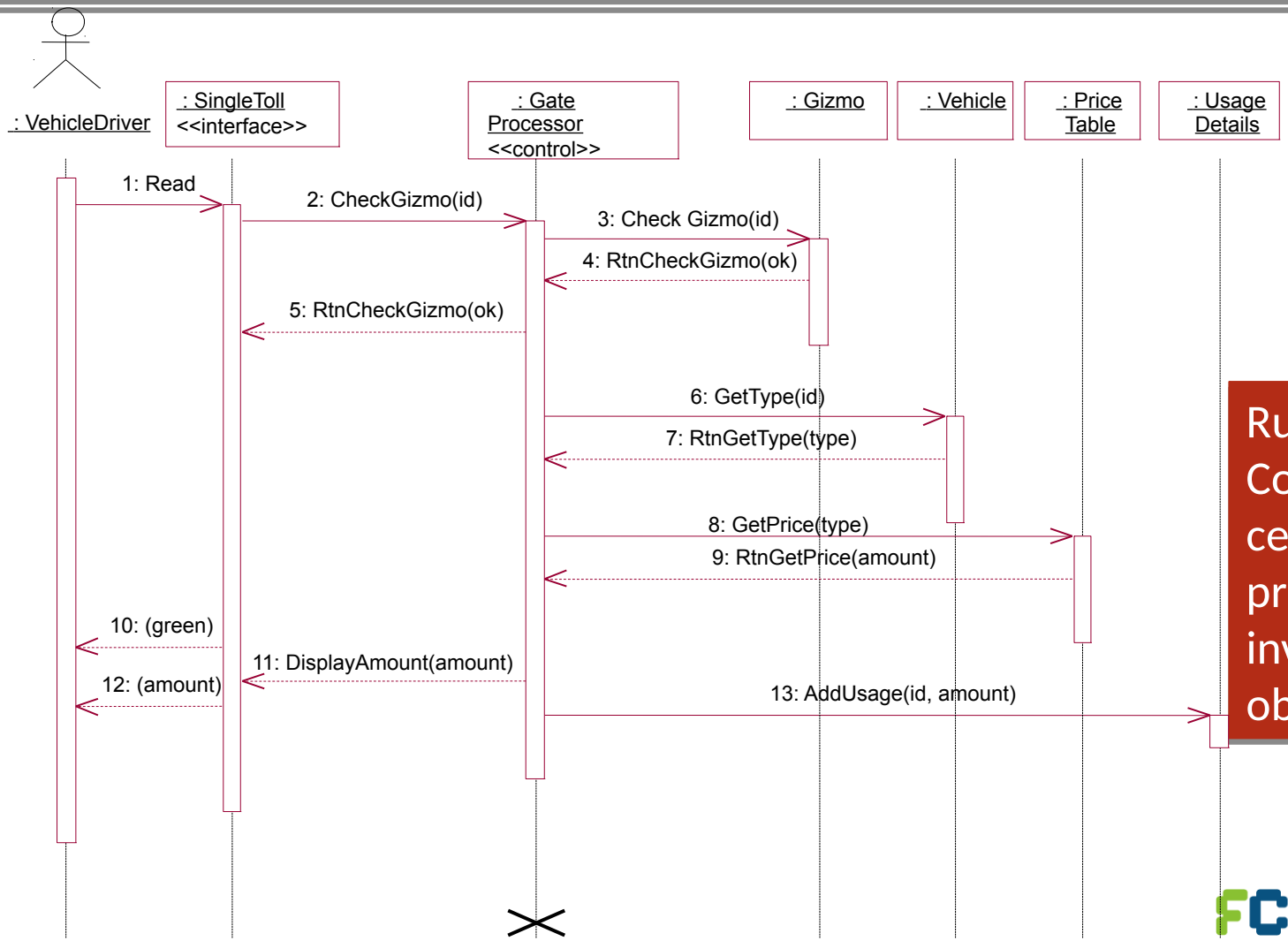
Rule Number 2: All messages from/to actors pass by the interface object.

with interface objects and control



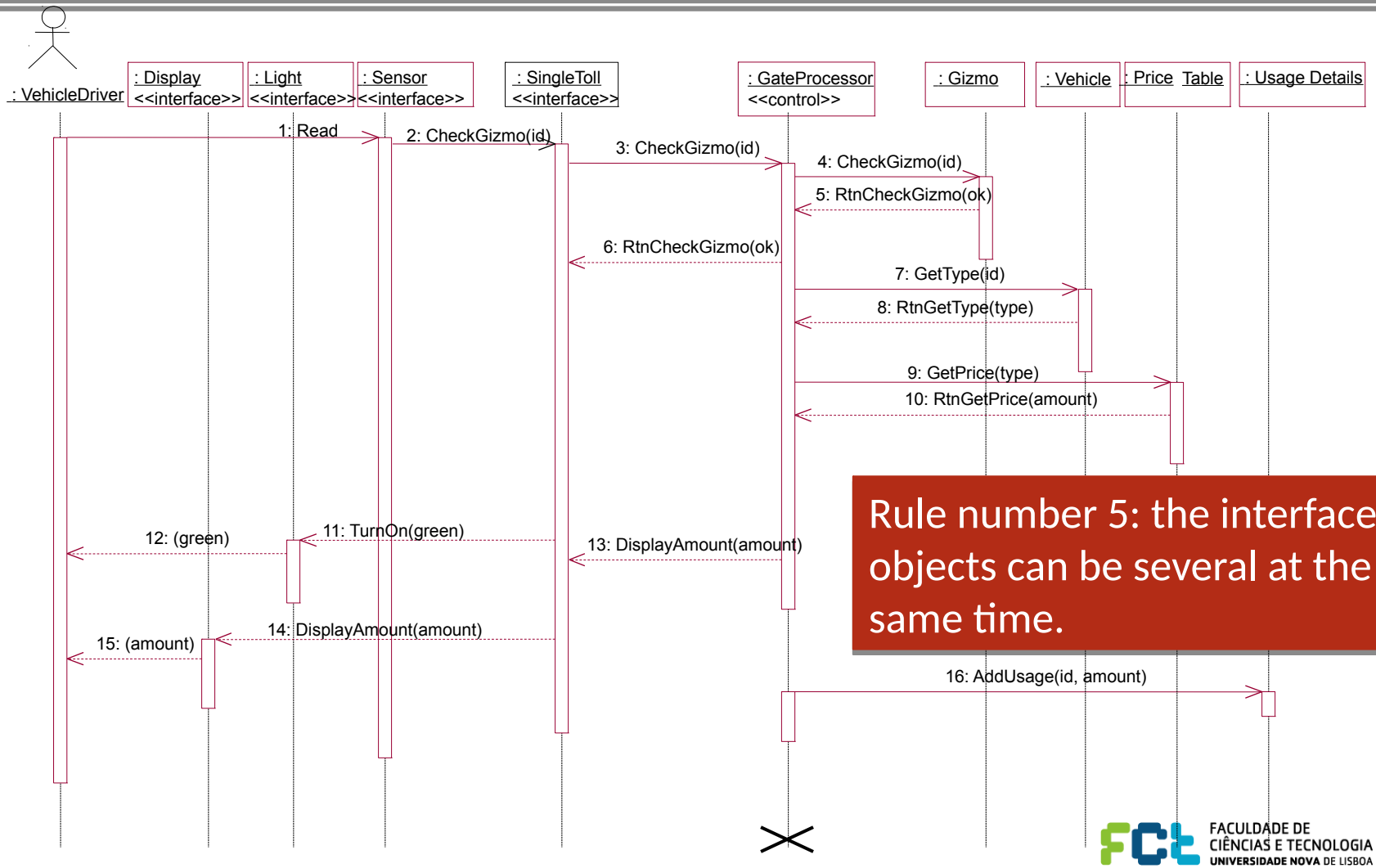
Rule number 3: we need control objects for complex functionalities.

with interface, control and entity objects



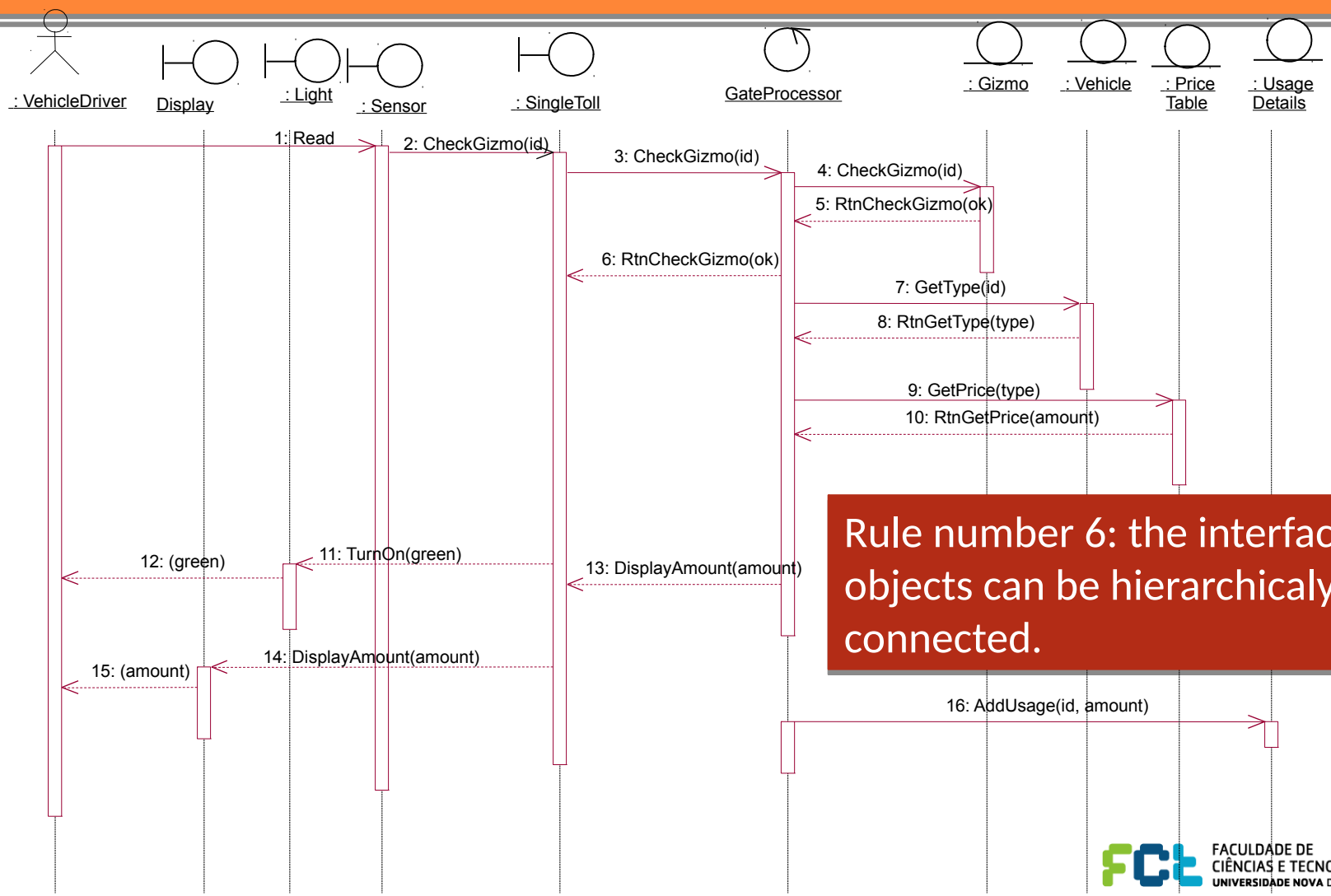
Rule number 4: Control objects centralize processing involving entity objects.

with the toll components



Rule number 5: the interface objects can be several at the same time.

Alternative notation



Rule number 6: the interface objects can be hierarchicaly connected.

Métodos de Desenvolvimento de Software (MDS) 2016/2017

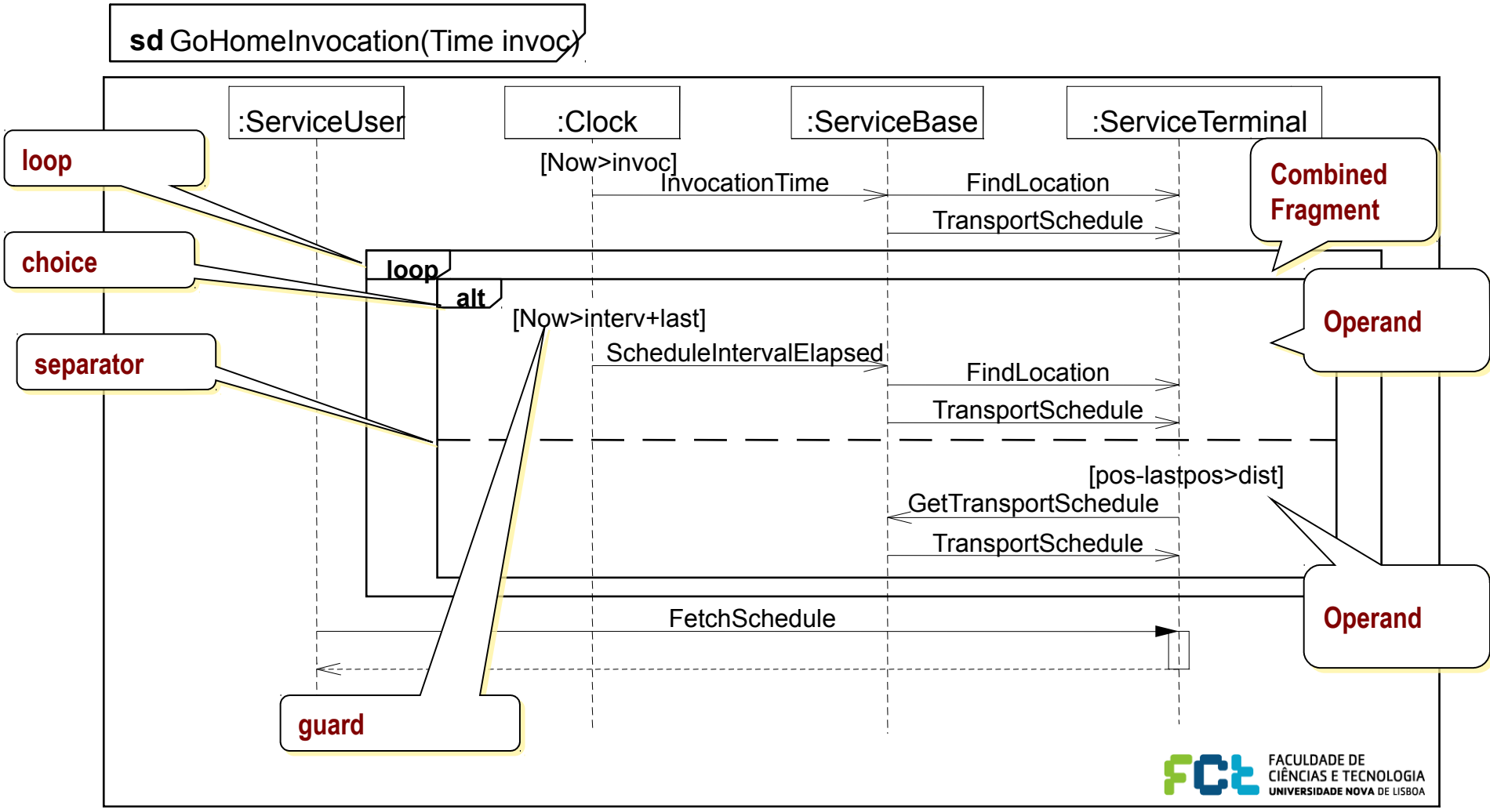
Interaction Diagrams

Combined Fragments and Operators

Combined Fragments and Operators

- Combined Fragments divide the interaction diagram in different areas with different behavior
- Composed by
  - One Operator
    - Determines **how** the operands execute
  - One or more operands
  - Zero or more guard conditions
    - Determine whether **if** the operands execute or not

Combined Fragments and Operators

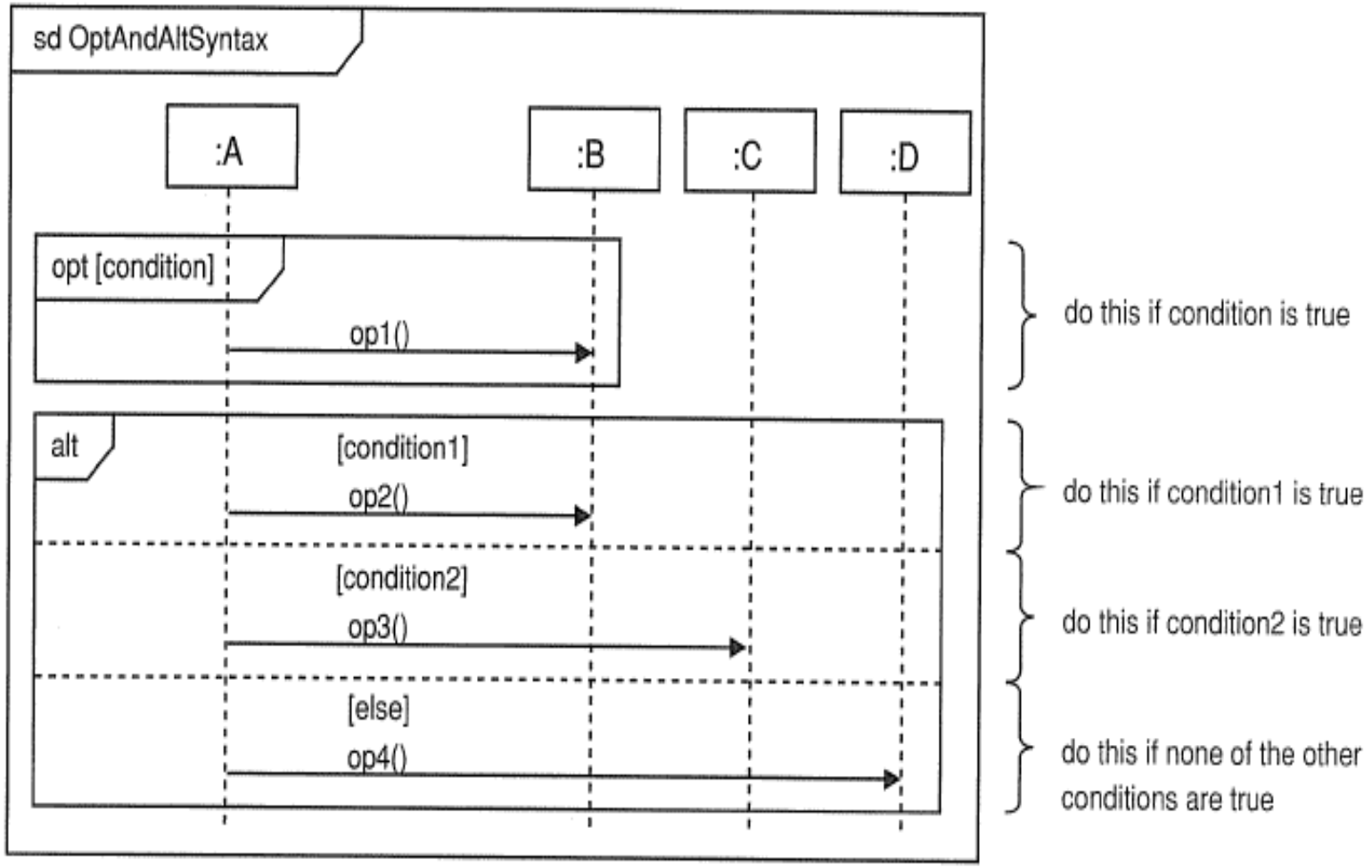


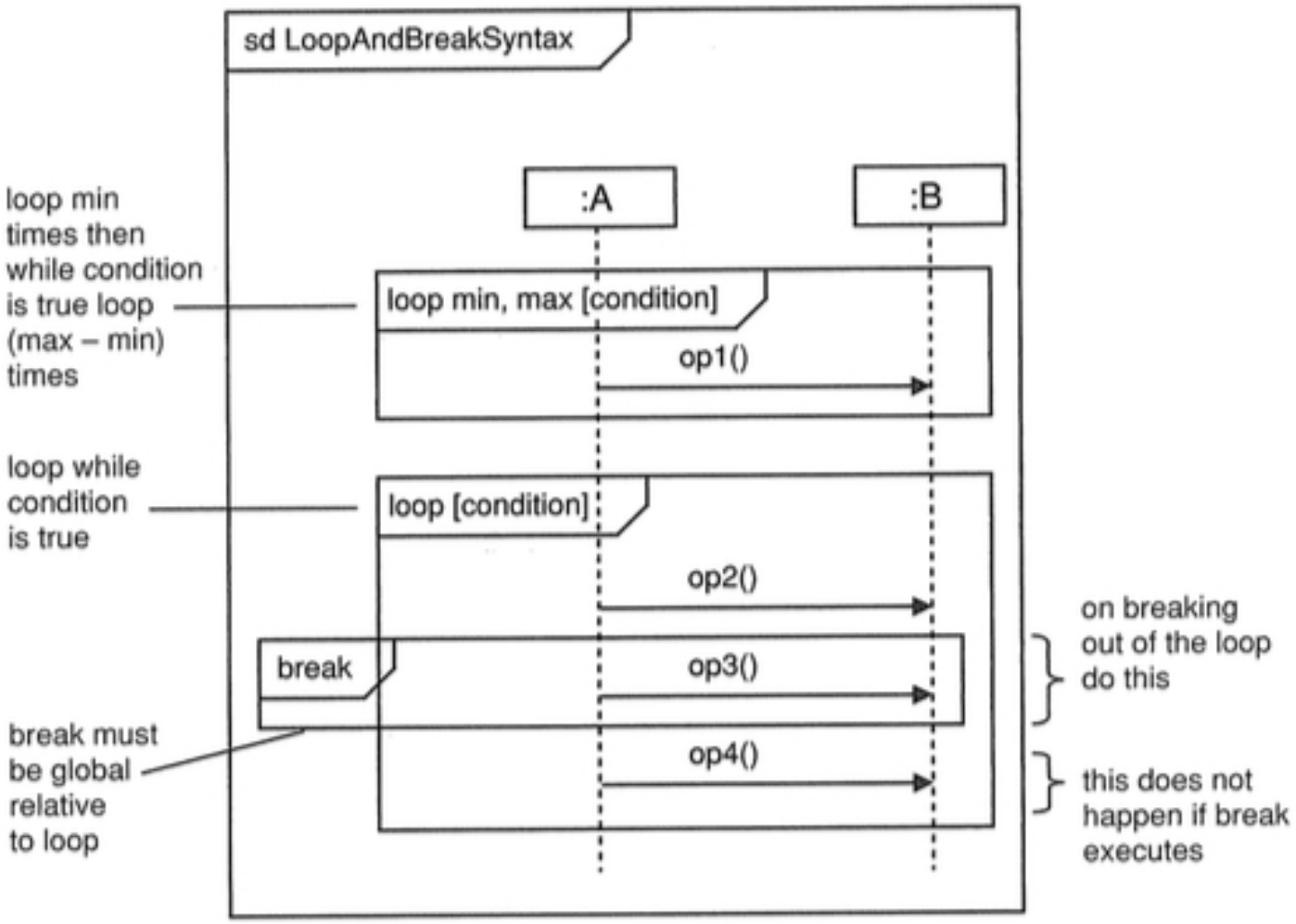
Types of Combined Fragments

- Alternatives (**alt**)
  - Alternative choice of Behavior – one at most is going to be executed
  - Depends on the guard (supports the “else” guard)
    - (similar to a switch)

- Option (**opt**)
  - Special alternative case where only one operand executes
    - (similar to an if… then)

- Interruption (**break**)
  - Represents one alternative that is executed instead of the rest of the fragment
    - (similar to a break in a cycle)
    - When the guard is true the operand executes, but not the rest of the interaction where it is inserted, the enclosing interaction should be terminated.
      - it is similar to if (guard){…; return;}


sd OptAndAltSyntax
:A
:B
:C
:D
opt [condition]
op1()
alt
[condition1]
op2()
[condition2]
op3()
[else]
op4()
do this if condition is true
do this if condition1 is true
do this if condition2 is true
do this if none of the other conditions are true


sd LoopAndBreakSyntax
:A
:B
loop min times then while condition is true loop (max – min) times
loop min, max [condition]
op1()
loop while condition is true
loop [condition]
op2()
break
op3()
op4()
on breaking out of the loop do this
this does not happen if break executes
break must be global relative to loop

Types of Combined Fragments

- Cycle (**loop**)
  - Optional Guard: [<min>, <max>, <Boolean-expression>]
    - loop min, max [condition]
      - loop min times, after, while condition is true, executes (max – min) times
  - The absence of a guard means that there is no specified limit (like *)
- Reference (**ref**)
  - Reference to other interaction
- Parallel (**par**)
  - All the operands execute interleaved. However the UML specifies that the interleaving of the event occurrences of the operands must be done in such a way that the ordering in the original operand is maintained

Types of Combined Fragments

- Critical (**critical**)
  - The traces can not be interleaved with events of other lifelines, which means that they execute without interruption
- Weak Sequencing (**seq**)
  - The operands execute in parallel in the different lifelines, with the following restriction: received events in the same lifeline, created by different operands occurr in the same sequence as the operands
- Strong Sequence(**strict**)
  - The Operands are executed in strict sequence

Types of Combined Fragments

- Negative (**neg**)
  - Identifies sequences that can not occur
- Ignore (**ignore**)
  - Lists messages not shown on purpose
    - Example: ignore {message1, message2, message3}
- Consider (**consider**)
  - Lists messages intentionally included
    - Example: consider {authenticateUser, sendEvelope, sendBody, disconnect, shutdown}
- Assertion (**assert**)
  - Represents the only valid behavior in a given interaction point

From Activities Diagrams toInteraction Diagrams

- Activity -> Class operation
- Transition -> Message
- Branching-Merge -> Alt
- Fork-Join -> Par
- (Cycle: Backwards transition) -> Loop

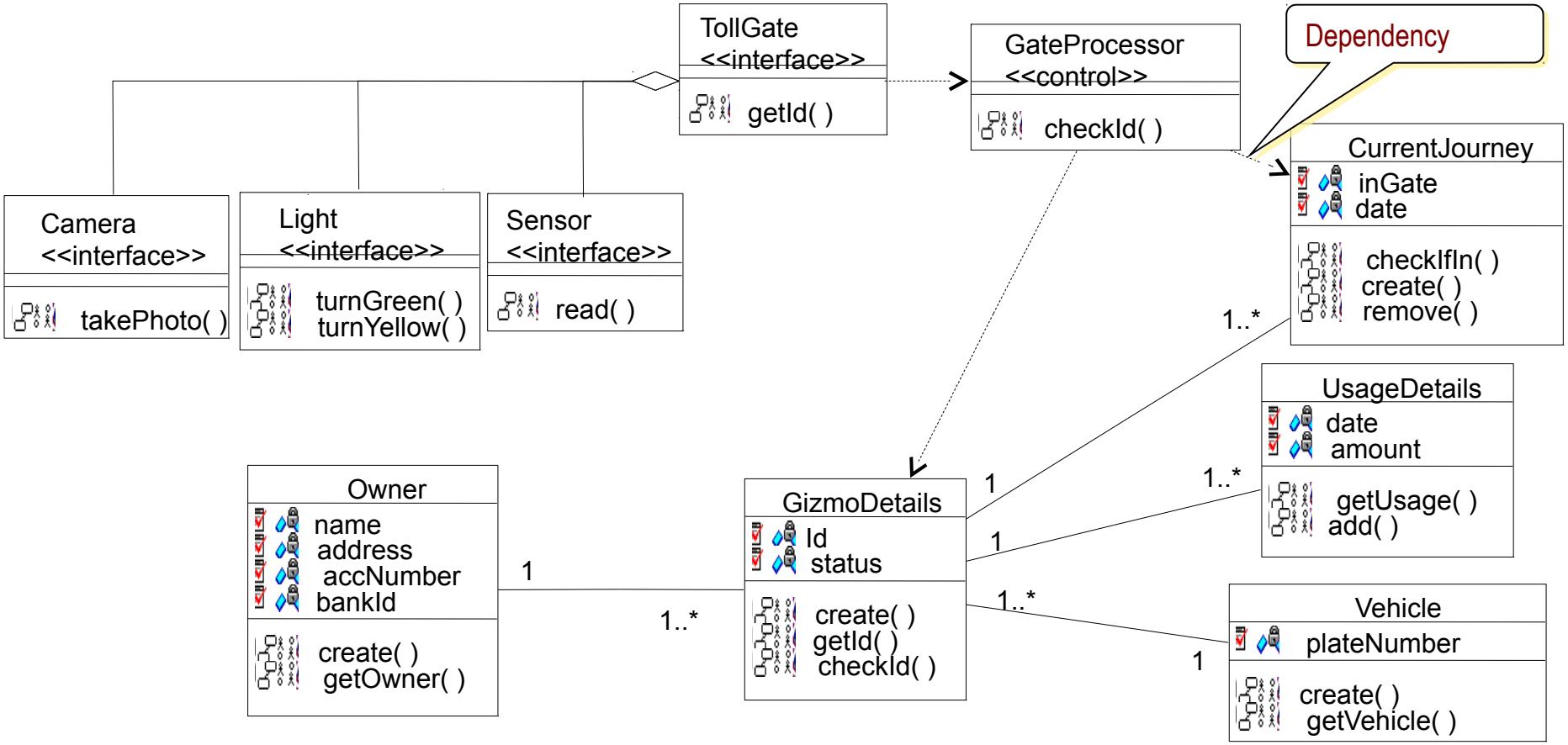
Updating the Class Diagrams

From Activities Diagrams to Interaction Diagrams

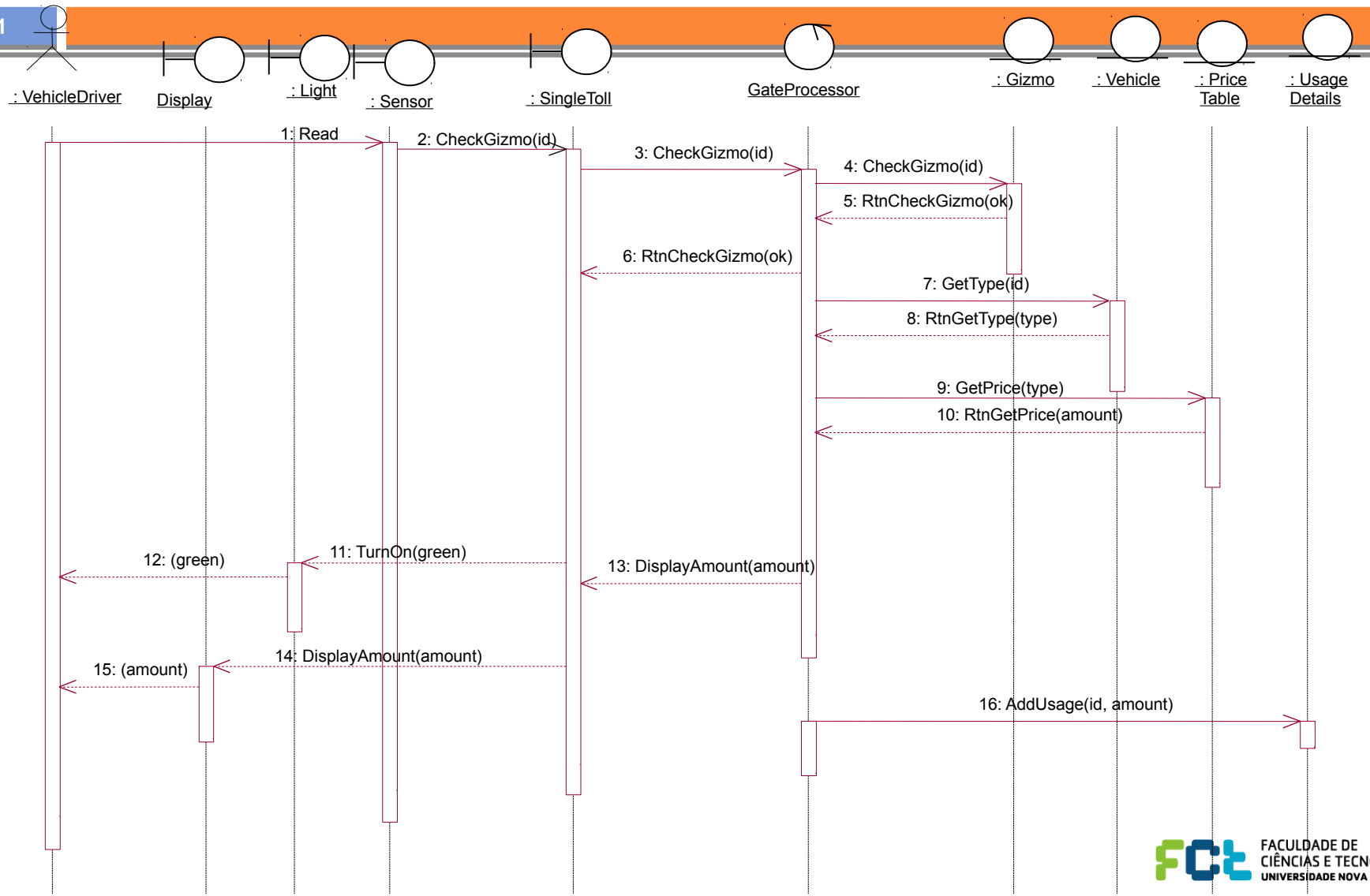
- The Domain Class Diagram must be completed with the information of the Sequence Diagrams
- In particular:
  - Lifeline -> Class
    - Interface, control, (Entity)
  - Message -> Operation
  - Message and arguments -> Association
  - Message -> Dependency

Partial Resulting Class Diagram

Example: the car passes by a toll one way (e.g. Bridge 25 the Abril)



Interaction Diagram



Communication Diagrams

Former Collaboration Diagrams in UML 1.0

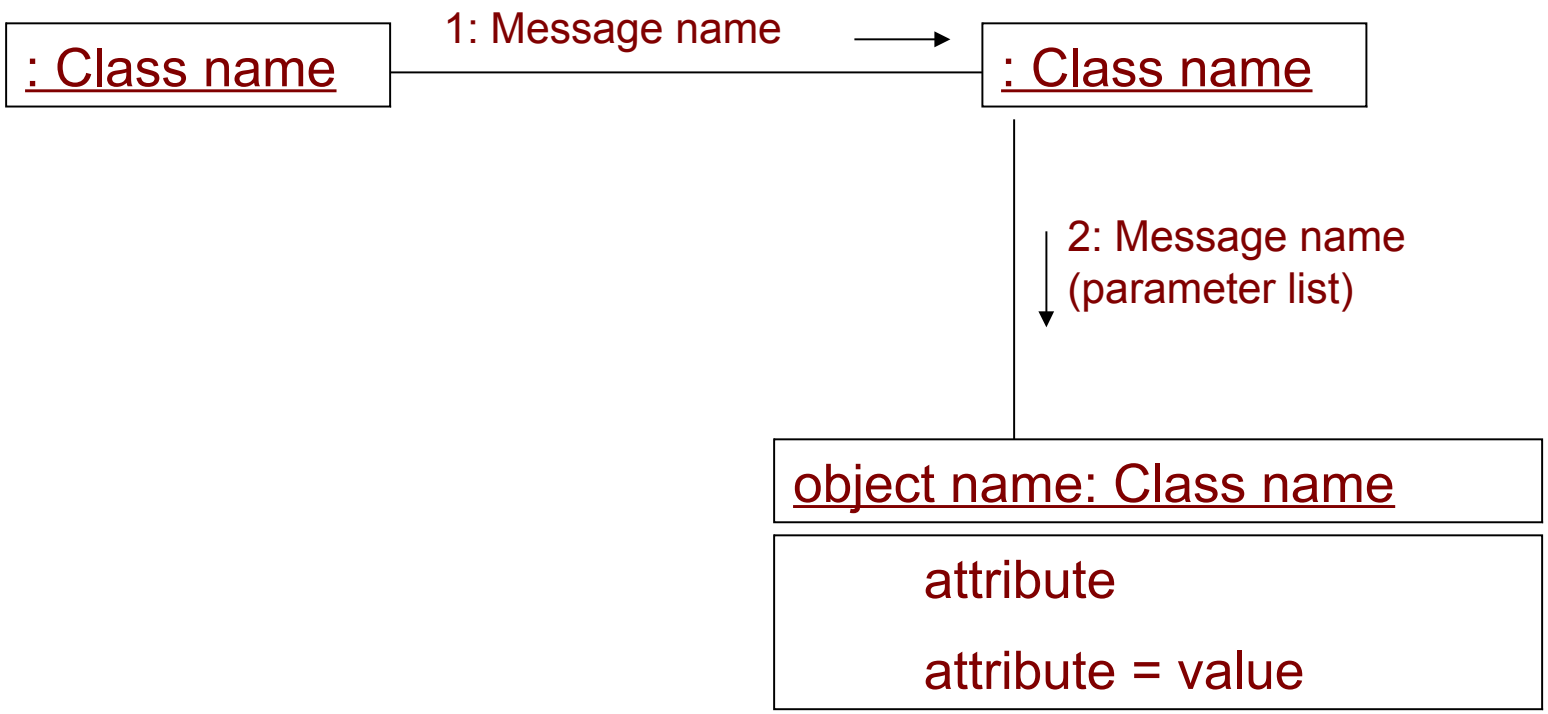
Communication Diagrams

- A communication diagram is an interaction diagram that stresses the structural organization of the objects that send and receive messages.
- Shows the set of objects, links in between these objects, and messages sent and received by these same objects.
- It is useful to illustrate the dynamic view of the system

Communication Diagram

- Shows a second way of expressing the sequence in between events
- The objects are shown in linked rectangles connected by lines that show links in between them
  - The numbers show the order in which the operations are executed
  - The numbers are written together with the names of the messages, and an arrow shows the way of the flow

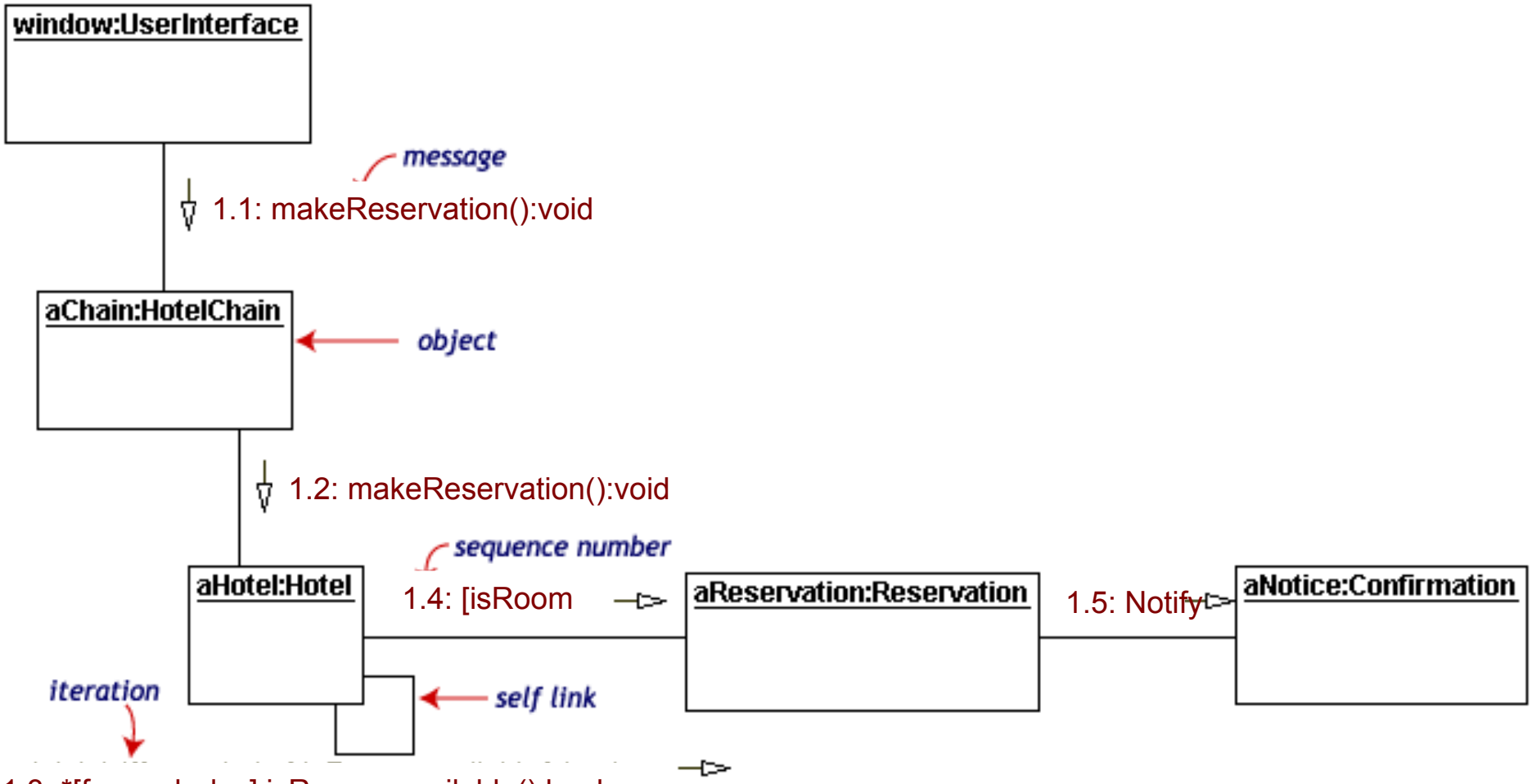
Communication Diagram: Simple example



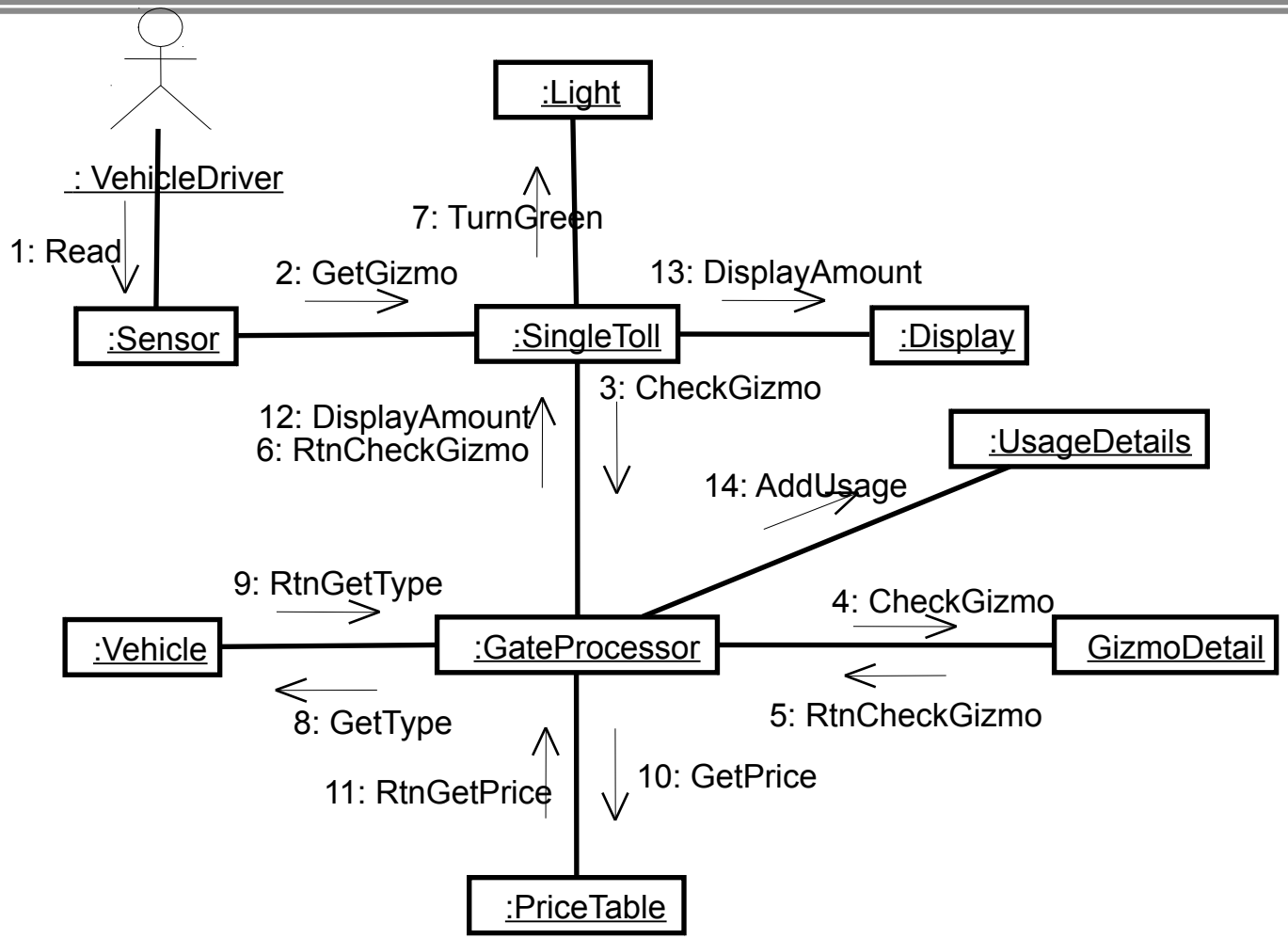
Communication Diagrams

- The numbering associated with the messages can represent nesting
  - E.g. 1.1 is the 1st message nested n message 1, 1.2 is the second and so on…
- Semantic equivalence with the Interaction Diagrams: can convert automatically a Interaction Diagram into a Communication Diagram

Communication Diagrams with message nesting



Communication diagram



Communication vs Interaction

- Communication Diagrams
  - Easy to rad and understand
  - The message order is very clear and intuitive
  - Show proper structures for cycles, concurrency, alternatives, etc
  - Require discipline while being constructed
- Communication diagrams
  - Less demanding regarding discipline while constructing(we don't know the exact place where the object will stay)
  - The message order can be added later
  - ...but are poorer