Inżynieria Oprogramowania / Inżynieria Systemów Informatycznych

dr inż. Michał Okulewicz

Wydział Matematyki i Nauk Informacyjnych

Projekt „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca” współfinansowany jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.
Zadanie 10 pn. „Modyfikacja programów studiów na kierunkach prowadzonych przez Wydział Matematyki i Nauk Informacyjnych”, realizowane w ramach projektu „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca”, współfinansowanego jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Software Engineering Exercises

Maciej Bednarz Michał Okulewicz
Mateusz Zaborski

February 2021

Contents

1 User Stories

User Stories are a tool for gathering requirements typically associated with Agile methodologies. They focus on the needs of future users, trying to capture the business justification behind each feature. A successfully list of User Stories is prioritized and the stories (just before implementing them) must be in the “ready” state.

1.1 Syntax

A single User Story consists of a front and reverse side. Front side specifies:

- the role of the user
- required action / feature
- business justification of the requirement

Reverse side defines acceptance criteria and non-functional requirements related to that story. This criteria should be defined by a checklist (yes/no) questions (e.g Does the website support HTTPS? Is response time below 50ms?).

One of the methods of prioritising User Stories is called MoSCoW, which is an acronym for Must Have, Should Have, Could Have and Won’t Have requirements.

1.2 Examples

As a...	**I want to...**	**so that...**
Customer	Order a package pickup	I can return products
On-site operator	View the orders on map	I can assign locations to vehicles
Courier	See the customer's phone number	I can contact him

Table 1: A sample list of user stories for package delivery company

ID: 02.01 TITLE: Assigning pickup locations

As a...	**I want to...**	**so that...**
On-site operator	View the orders on map	I can assign locations to vehicles

Figure 1: Front side of a User Story card

Acceptance criteria checklist
(**F**unctionality)

- Can I assign location to any available vehicle?
- Can I reassign location to another vehicle?
- Can I preview order details at given location?
- Can I observe the expected route and vehicle load?

Non Functional Requrements
(**U**sability **R**eliability **P**erformance **S**ecurity)

- Are the locations color coded (by vehicle)?
- Is the new route computed in less than 5 seconds?
- Can other on-site operators change my assignment?

Figure 2: Reverse side of a User Story card

Table 1 presents an example of User Stories list.

1.3 Case Study

You are about to gather requirements for a studies management system (e.g. USOS). Its main goal is to support faculty members, students and administration in organizing, conducting and grading particular courses.

The requirements for the system will be specified by the following stakeholders:

Vice Dean - you are responsible for maximal utilization of the faculty members with regards to their availability and costs, while maintaining a consistent program of studies across whole program.

Administrative Worker - you are responsible for managing all non-standard cases regarding passing the subjects and special needs of the students (impairments, parallel studies, individual studies, leave of absence, foreign and domestic exchange)

Schedule Coordinator - you are responsible for assigning rooms, time slots and teachers for all conducted classes.

Senior Faculty - you are responsible for giving lectures, examining students, maintaining consistency between lab supervisors, and apart from that competing for research funds

Junior Faculty - you are responsible for conducting classes, grading tests, preparing exercises and exams, and apart from that trying to conduct research

Teachers: see Appendix A for hints.

1.3.1 Exercise 1

Work in groups of (around) 5. Assume one of the following roles: Vice Dean, Administrative Worker, Schedule Coordinator, Senior Faculty (professor, subject supervisor), Junior Faculty (PhD, labs supervisor). Prepare at least 3 user stories per each role with regards to the studies management system.

1.3.2 Exercise 2

Choose two people who will now act as part of project/development team. Your goal is to negotiate priorities of created user stories according to the following rules:

Must Have - without this requirements the system has no sense at all, and is unsafe to use (e.g. allows for assigning multiple classes to a single student at the same time without any warning)

Should Have - it would be REALLY annoying not to have those requirements, however the system will be usable (e.g. there is no possibility of editing records. However, you may delete the old one, and create a new one in its place)

Could Have - every other feature which will be nice to have in the system, but definitely it is not a crucial one

Won't Have - features which seem to be out of scope of the project (at least for now)

1.3.3 Exercise 3

Select 3 requirements with top priority, and create the reverse side of the User Story card for them.

2 UML Use Case Diagrams

Use Case Diagrams are used for definition the requirements that system must fulfill. Their purpose is similar to the purpose of the User Stories. However, their structure is more formal and provides more information. They present actors, expected functionalities but also some dependencies between them.

2.1 Syntax

A single Use Case Diagram includes the following:

- system boundary
- actors
- use cases
- relationships between actors (generalization)
- relationships between use cases
  - ≪ include ≫
  - ≪ extend ≫
- associations

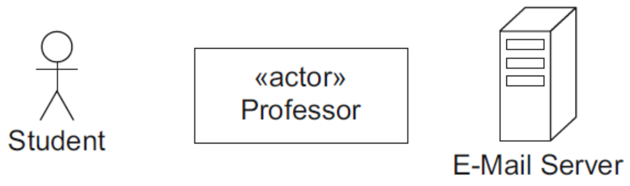


Figure 3: Notation alternatives for actors[1]

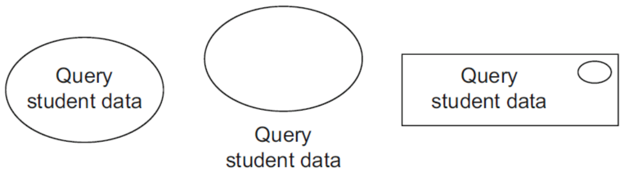


Figure 4: Notation alternatives for use cases[1]

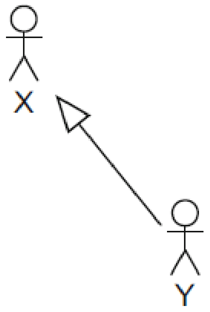


Figure 5: Example of generalization (inheritance) for actors[1]

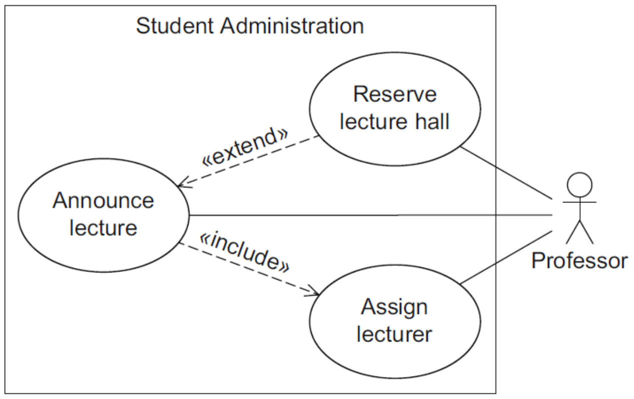


Figure 6: Relations (extend and include) between use cases[1]

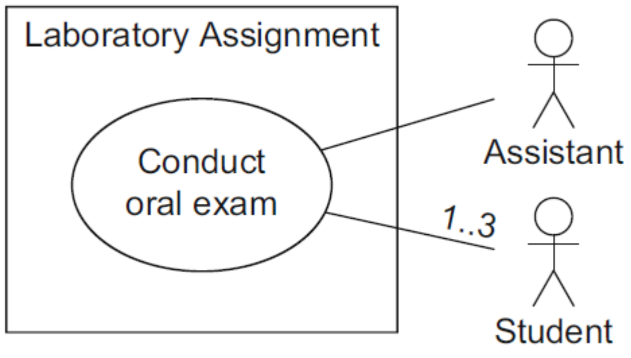


Figure 7: Multiplicities in associations[1]

2.2 Examples

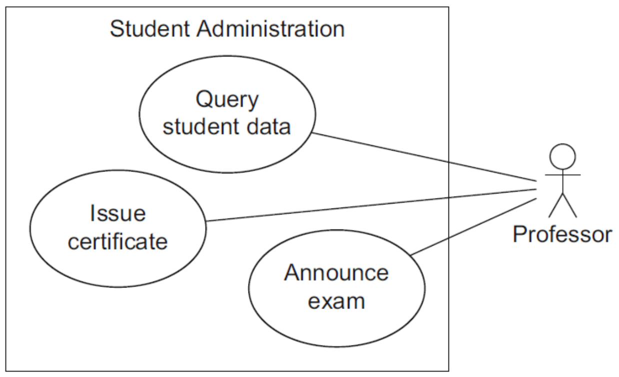


Figure 8: Example of a Use Case Diagram[1]

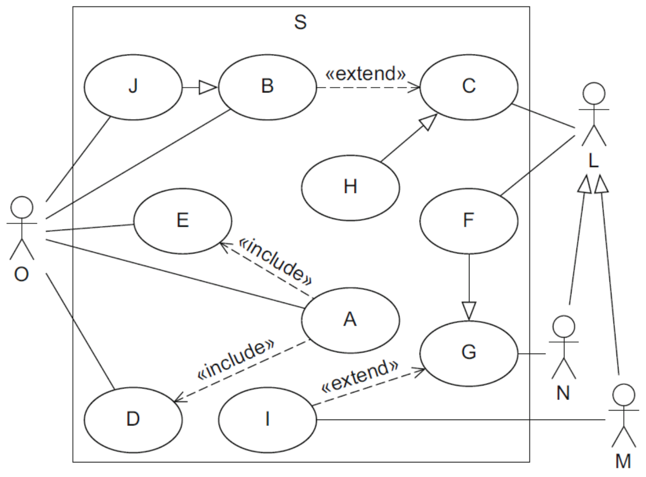


Figure 9: Example of relationships in a Use Case Diagram[1]

2.2.1 Exercise 1

Think about system for car sharing company. Create simple Use Case Diagram containing two actors: customer and service. Consider following customer actions: find on the map, choose a car, rent, refuel and park. The service is authorized to block and unblock a car.

2.2.2 Exercise 2

Please extend your diagram with extra use cases. What kind of action is included in refueling? What are extensions of the other use cases? Maybe reporting of a damage or sending a photo?

2.2.3 Exercise 3

Choose one use case from proposed Use Case Diagram and describe it according to the example on p. 36 from UML@Classroom[1].

3 UML Class Diagrams

UML Class Diagram are one of the structural diagrams (other examples include package or deployment diagrams). The purpose of this diagrams is to design the overall architecture of the system and depict possible entanglements of the already written code.

UML Class Diagrams enable to capture API of the application and present relation between objects.

3.1 Syntax

The basic purpose is to present the fields (properties) and methods of a given class. Access levels are depicted with the usage of "+", "#", "∼" and "-" symbols before the name. Field type or method return type are given after a colon (you may use C or Java language types as commonly recognizable). An example of single class diagram is depicted in Figure 10.

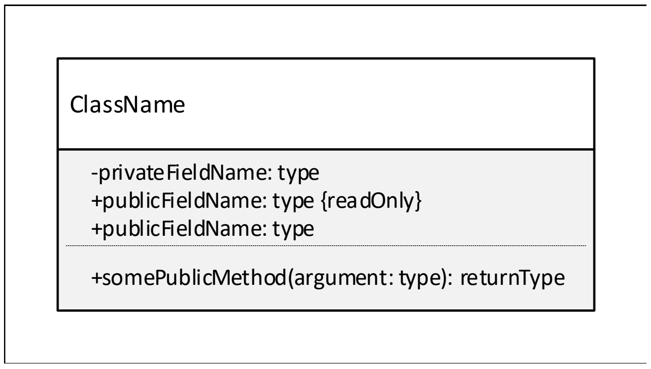

ClassName
-privateFieldName: type
+publicFieldName: type {readOnly}
+publicFieldName: type
+somePublicMethod(argument: type): returnType

Figure 10: Single class example

In order to keep the diagrams simple you can stick just to the publicly accessible fields and methods as a way of discussing the API, while private fields are implementation details which will be designed "on-the-fly" anyway.

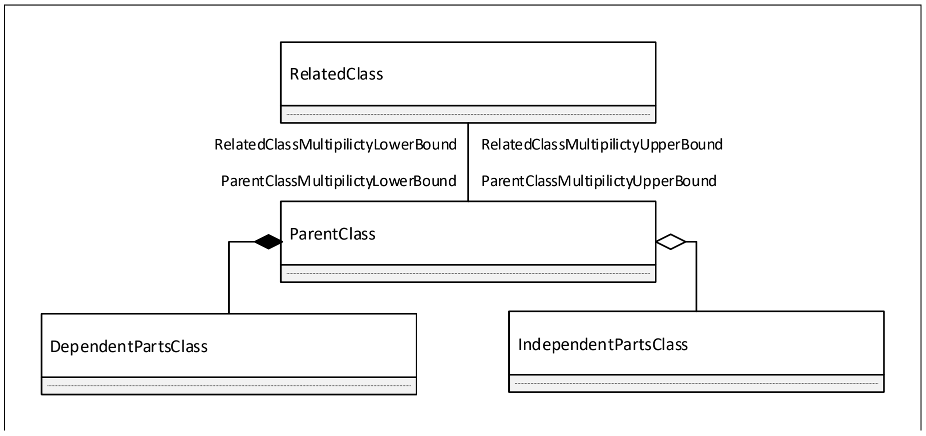

RelatedClass
RelatedClassMultipilictyLowerBound
RelatedClassMultipilictyUpperBound
ParentClassMultipilictyLowerBound
ParentClassMultipilictyUpperBound
ParentClass
DependentPartsClass
IndependentPartsClass

Figure 11: Class relations example

Another thing that can be captured in the class diagrams are relation between classes (meaning that a class references another or includes a collection of objects of another class). When this relation is symmetrical (the objects are of equal "hierarchy") you can use a simple line, depicting association. When there is some kind of parent-child relation you can use aggregation (empty diamond ending), or composition (full diamond ending). The subtle difference between those relations is in the fact whether elements of child class have a meaningful existence without the parent class (aggregation) or not (composition). An example of the syntax related to the references between classes is presented in Fig. 11.

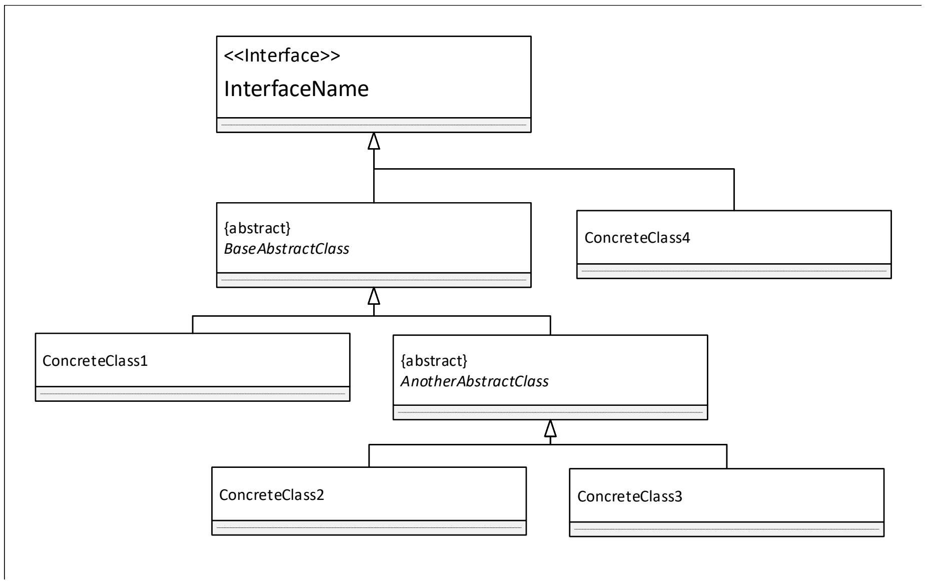


Figure 12: Inheritance and interface implementation

Finally, Figure 12 presents how to depict interface implementation and inheritance structure.

For more details on UML Class Diagrams please checkout Chapter 4 from UML@Classroom[1].

3.2 Examples

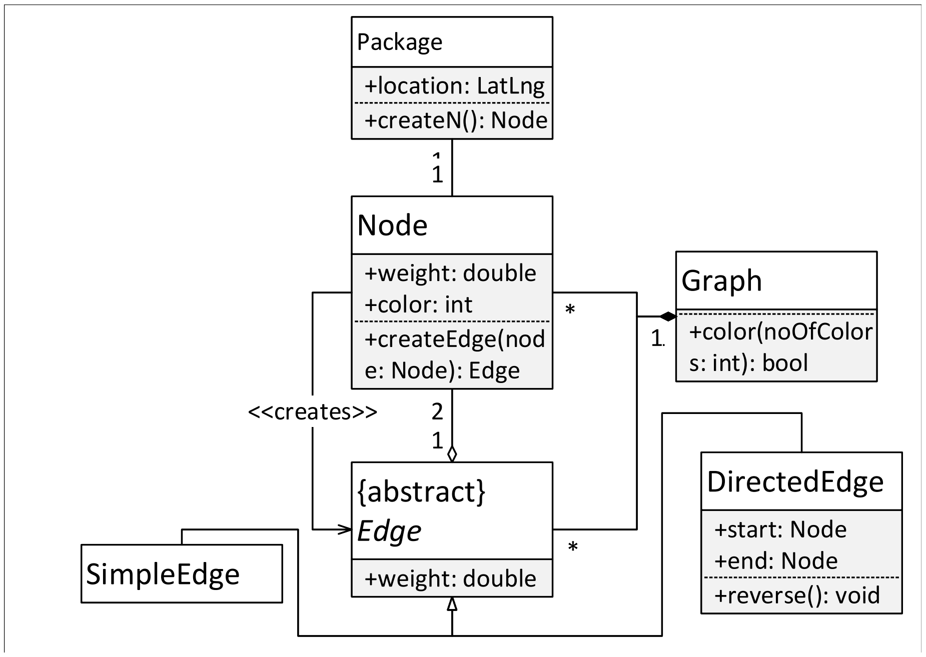


Figure 13: An example of a graph representation

Figure 13 gives an example of relations between Graph and its elements: Nodes and Edges. Such an abstract model might be used to represent a business problem of package delivery, hence the example of associating a Node with a Package.

3.3 Case Study

Consider a Real Time Strategy game with different types of terrain and different types of units.

Terrain types:

- Road
- Ground
- Jungle
- Desert
- Mountains
- Rivers

Unit types:

- Walking units
  - Soldier
  - Medic
- Vehicle units
  - Jeep
  - Tank
- Flying units
  - Fighter
  - Bomber

For teachers: see Appendix B for hints.

3.3.1 Exercise 1

Create a UML Class Diagram for unit types and terrain types. Propose methods for verifying possibility of attack and ability to move over certain type of terrain. Design this with the assumption that terrain types does not change as much as unit types (with future game extension).

3.3.2 Exercise 2

Enhance class diagram with the environment (map) where units interact with one another and the terrain.

3.3.3 Exercise 3

Add buildings for producing units.

4 UML State Diagrams

The Stale Machine Diagrams are a a tool for modeling the possible states for the system or the object. Furthermore they show how state transitions occur as a consequence of occurring events, and what behavior the system or object exhibits in each state

4.1 Syntax

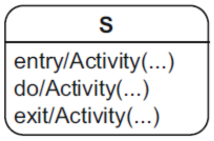
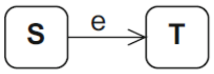
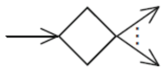
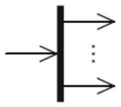
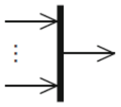
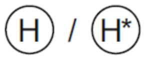
Name	*Notation*	*Description*
State	 S entry/Activity(...) do/Activity(...) exit/Activity(...)	Description of a specific "time span" in which an object finds itself during its "life cycle". Within a state, activities can be executed on the object.
Transition		State transition e from a source state S to a target state T
Initial state		Start of a state machine diagram
Final state		End of a state machine diagram
Terminate node		Termination of an object's state machine diagram
Decision node		Node from which multiple alternative transitions can proceed
Parallelization node		Splitting of a transition into multiple parallel transitions
Synchronization node		Merging of multiple parallel transitions into one transition
Shallow and deep history state		"Return address" to a substate or a nested substate of a composite state

Figure 14: Notation elements for the state machine diagram[1]

4.2 Examples

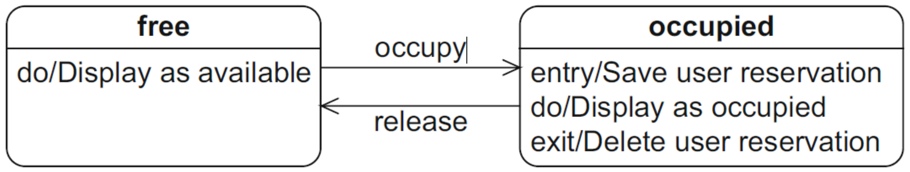


Figure 15: Example of state machine diagram of a lecture hall with internal activities [1]

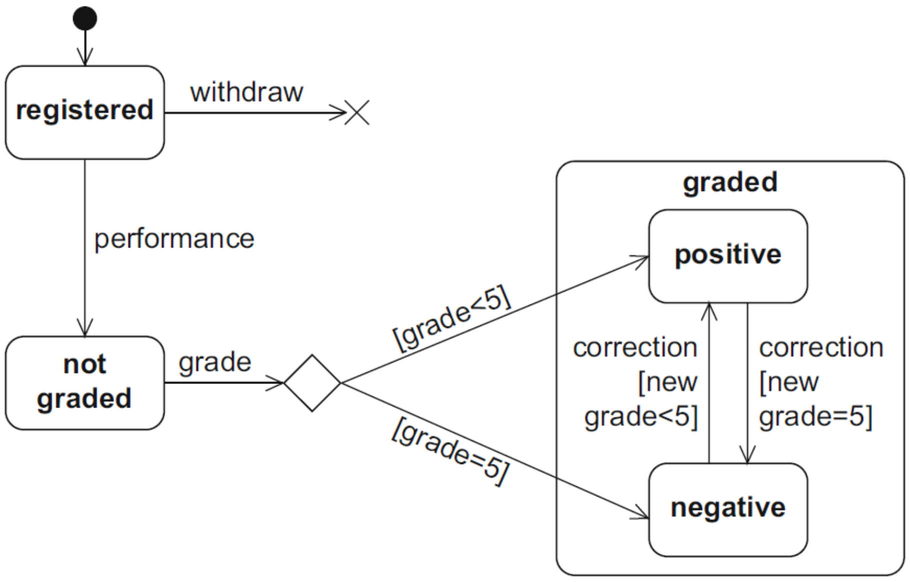


Figure 16: Example of state machine diagram of of a student's course participation [1]

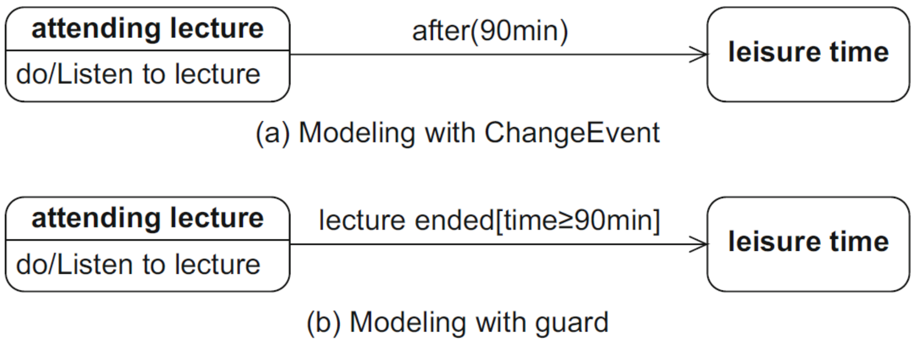


Figure 17: Time triggers in state machine diagrams [1]

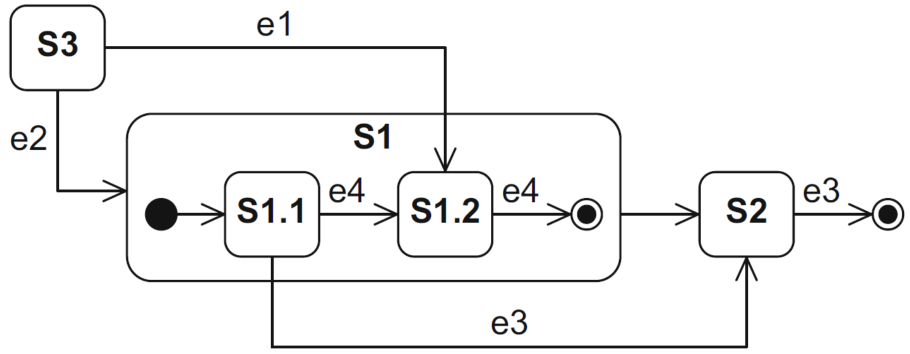


Figure 18: Composite state in state machine diagrams [1]

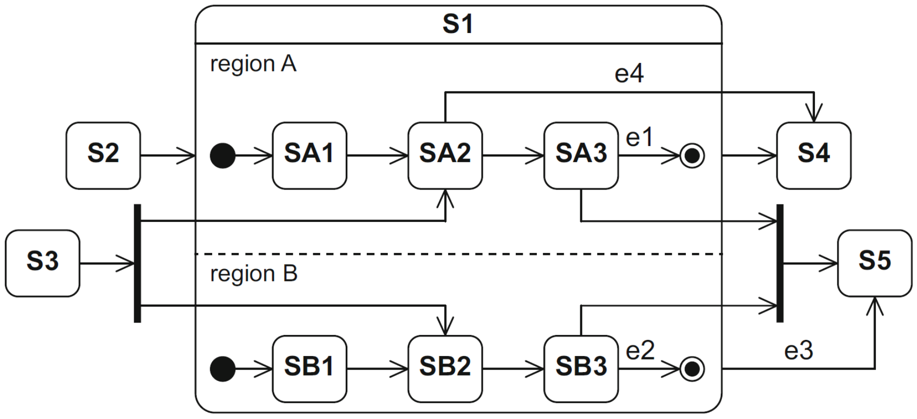


Figure 19: Orthogonal state in state machine diagrams [1]

4.3 Exercises

4.3.1 Exercise 1

Think about newsletter service. Please make a simple state diagram of customer object. Assume that customer can be subscribed, unsubscribed and deleted. Please include states and necessary actions.

4.3.2 Exercise 2

Extend above diagram by internal activities (entry/, do/, exit/).

4.3.3 Exercise 3

Consider ATM transaction processing. Please make a state diagram for transaction object. Consider following states: idle, card reading, PIN entry, verification, choosing transaction, performing transaction, ejecting card. Remember about maximum PIN trials. What kind of internal activities can be added?

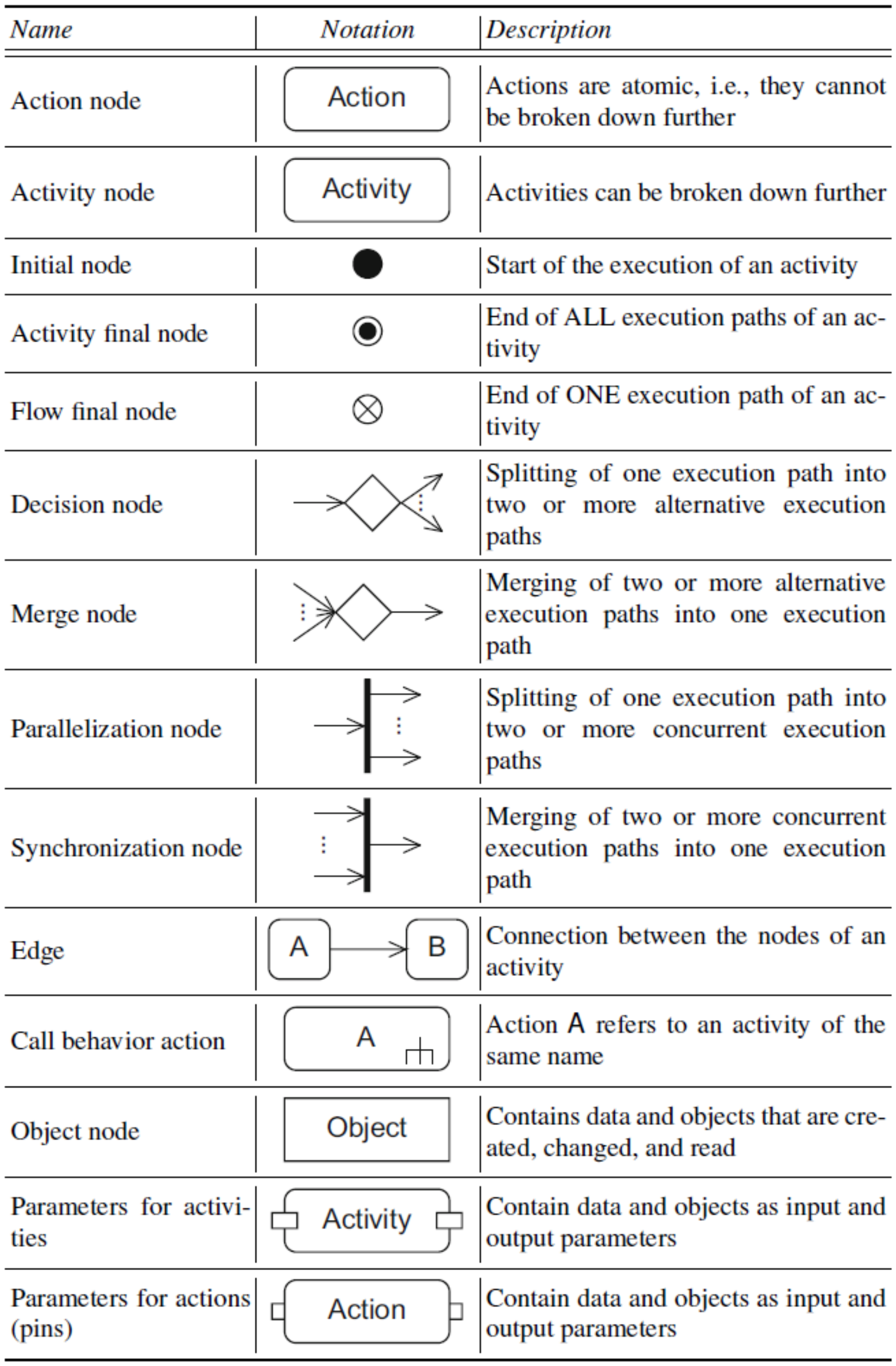
5 UML Activity Diagrams

UML Activity diagrams are a merge between flowcharts and Petri nets. They can be utilized to depict execution paths within a single method, a flow of activity through the system (say methods called when a system functionality is utilized) or an algorithm.

In connection with other diagrams it can be thought as

- an orthogonal view to State Diagrams
- in depth view of Use Case diagrams
- an addendum to Sequence Diagrams

5.1 Syntax



Name	*Notation*	*Description*
Action node	Action	Actions are atomic, i.e., they cannot be broken down further
Activity node	Activity	Activities can be broken down further
Initial node	●	Start of the execution of an activity
Activity final node	◉	End of ALL execution paths of an activity
Flow final node	⊗	End of ONE execution path of an activity
Decision node		Splitting of one execution path into two or more alternative execution paths
Merge node		Merging of two or more alternative execution paths into one execution path
Parallelization node		Splitting of one execution path into two or more concurrent execution paths
Synchronization node		Merging of two or more concurrent execution paths into one execution path
Edge	A → B	Connection between the nodes of an activity
Call behavior action	A	Action A refers to an activity of the same name
Object node	Object	Contains data and objects that are created, changed, and read
Parameters for activities	Activity	Contain data and objects as input and output parameters
Parameters for actions (pins)	Action	Contain data and objects as input and output parameters

Figure 20: Notation elements for the activity diagram[1]

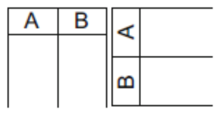
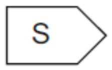
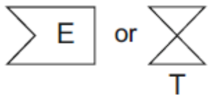
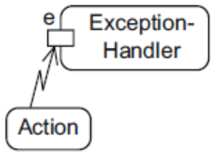
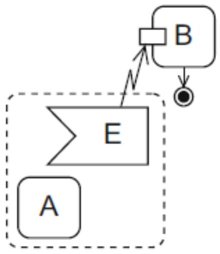
Name	*Notation*	*Description*
Partition		Grouping of nodes and edges within an activity
Send signal action		Transmission of a signal to a receiver
Asynchronous accept (time) event action		Wait for an event E or a time event T
Exception handler		Exception handler is executed instead of the action in the event of an error e
Interruptible activity region		Flow continues on a different path if event E is detected

Figure 21: Notation elements for the activity diagram[1]

5.2 Examples

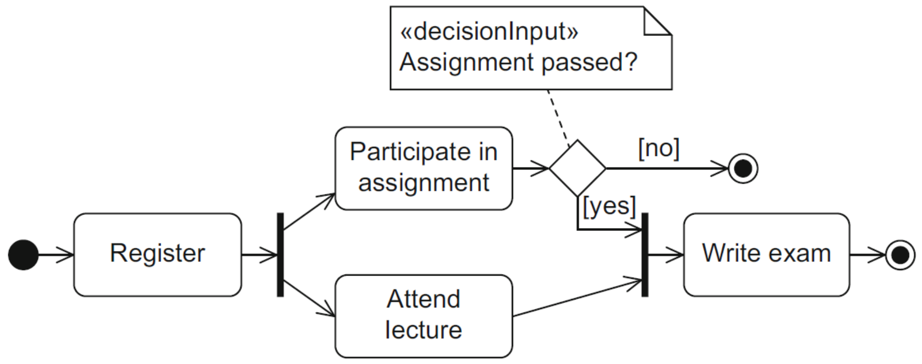


Figure 22: Example of activity diagram [1]

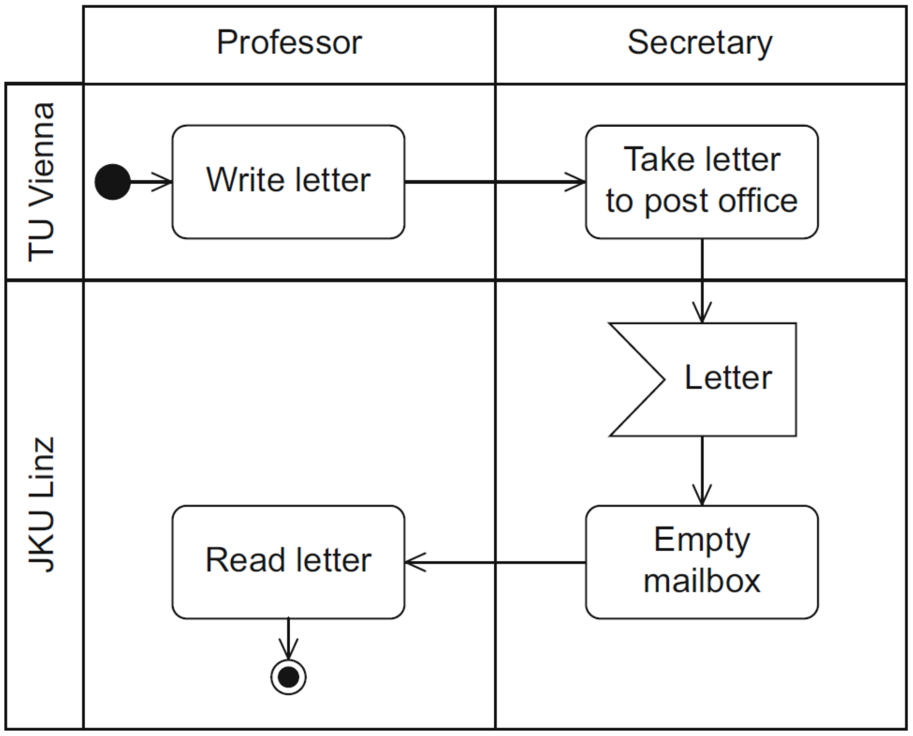


Figure 23: Example of activity diagram with partitions[1]

5.3 Exercises

5.3.1 ATM cash withdrawal

Think again about ATM machine from state diagrams (you may check Fig. 31 in the appendix for reference). Please create activity diagram for the cash withdrawal. What is the relation of this diagram to state diagram?

5.3.2 Coffee machine

Create the diagram of activity for a simple coffee machine, performing the actions in compliance with a description:

- self test
- parallel check if water and the coffee beans are available
- information printed on the screen that procedure has been started
- parallel water pressurisation and coffee grinding
- coffee brewing
- information printed on the screen that coffee is ready

When any resource is ended, a message should be printed on a screen and once it has been refilled the procedure is continued.

5.3.3 Extended coffee machine

Create or enrich the diagram created in a last exercise, taking into account that the following device components are taking the contribution in a coffee brewing process.

- pump (makes water pressure)
- coffee grinder (grinds the coffee)
- brewing block (brews the grinds coffee)
- milk frother (prepare foamed milk)
- screen (prints the messages)
- controller (do the other job)

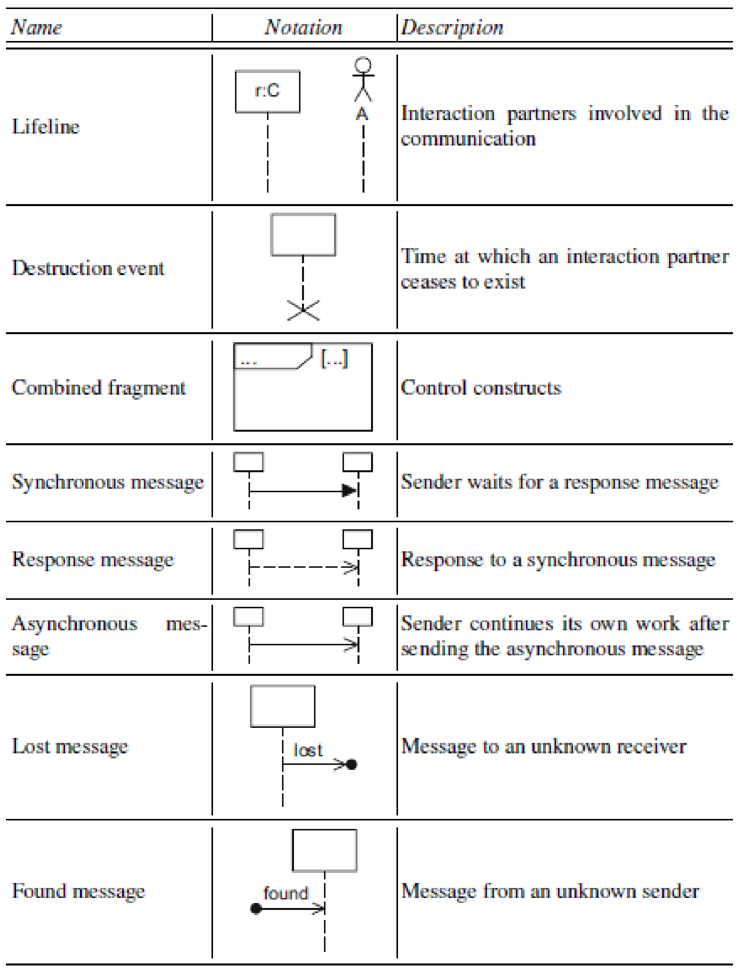
The components listed above should be also shown on the diagram. There should be added an additional action when (parallel with brewing the coffee), a milk foam is made.

6 UML Sequence Diagrams

The interaction diagrams provide important information: how the messages are exchanged between interaction partners (human and nonhuman beings like servers or a software) and what is the order of sending them. Thanks that we know when a one side of the interaction should receive a message and when messages A must not appear before a message C.

6.1 Syntax

The notation of the mostly used graphical elements is presented below.



Name	*Notation*	*Description*
Lifeline	r:C A	Interaction partners involved in the communication
Destruction event		Time at which an interaction partner ceases to exist
Combined fragment	... [...]	Control constructs
Synchronous message		Sender waits for a response message
Response message		Response to a synchronous message
Asynchronous message		Sender continues its own work after sending the asynchronous message
Lost message	lost	Message to an unknown receiver
Found message	found	Message from an unknown sender

Figure 24: Notation elements for the sequence diagram[1]

6.2 Examples

6.2.1 Example of alternatives

Consider the situation when a user (presented as an actor on the left side of the diagram) wants to buy any identifiable product (this, concrete instance of the product with an id no 157) and pay for it. But the money transfer may be done when once the concrete product will have been blocked (to avoid the situation when two or more customers pay for the same concrete item). Bear in mind that here the actor talks with a server API and is not interested on how the server will do it. He sends a request only and does not take care on synchronization with any other possible requests in the case. The alternatives covers the cases when something wrong was with a payment or a blocking process.

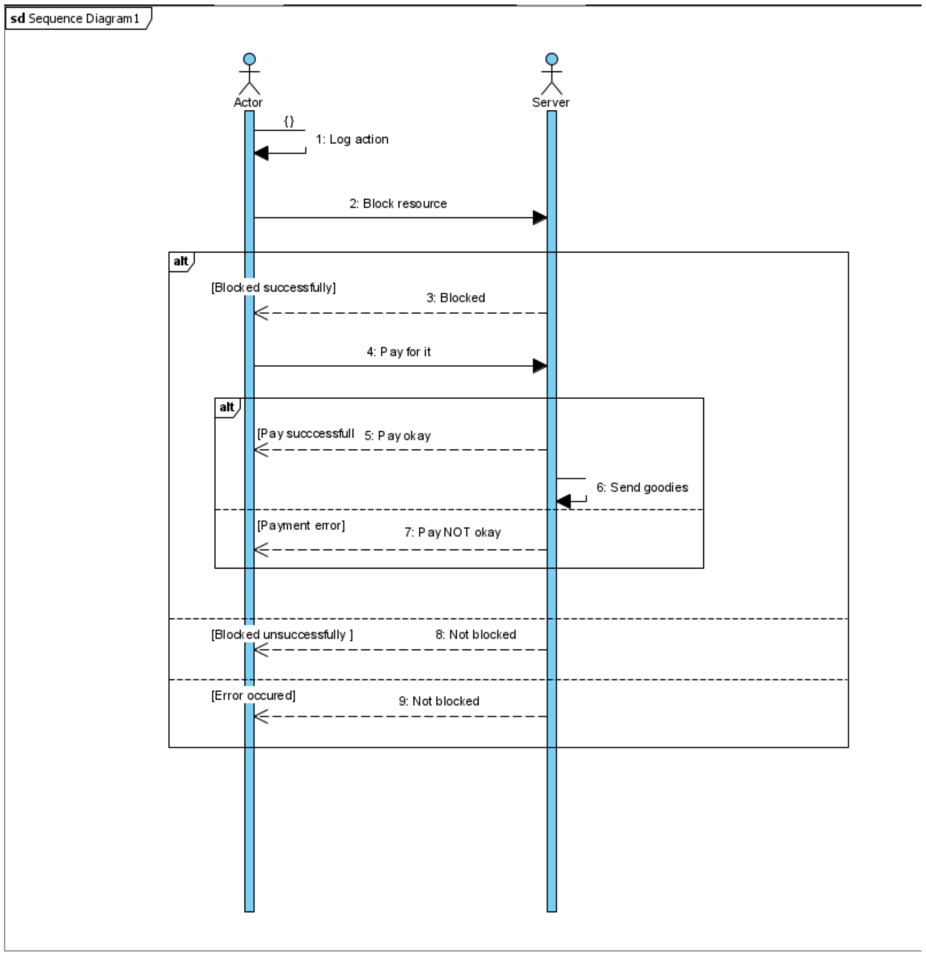


Figure 25: Notation elements for the sequence diagram describing the alternatives[1]

6.2.2 Example of loops and breaks

Think about a game machine. The user tries to guess the number three times. When hit the value, the information about the win is send to the player. Otherwise after three unsuccessful probes (a loop element) the flow is terminated (by a break element) and "game over" message is sent back.

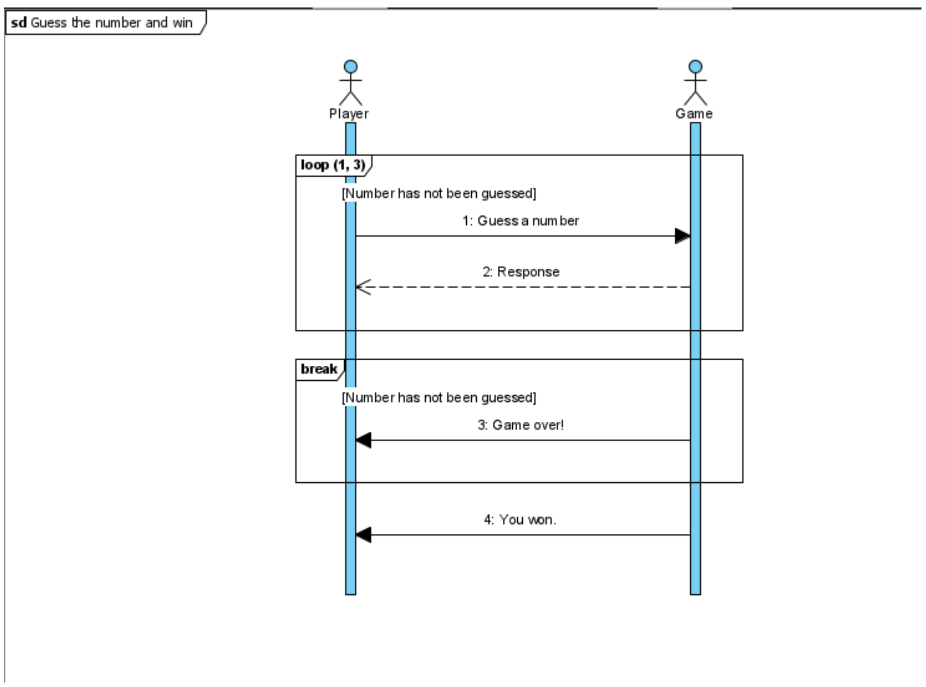

sd Guess the number and win
Player
Game
loop (1, 3)
[Number has not been guessed]
1: Guess a number
2: Response
break
[Number has not been guessed]
3: Game over!
4: You won.

Figure 26: Notation elements for the sequence diagram describing loops and breaks [1]

6.2.3 Example of parallel messages

Try to imagine a virtual system for voting. We have a box and a three voters among whom we can distinguish one more important (Main voter) and two less important (voter A and voter B). We have a three votes. During the first one it does not matter who vote as first. All voters may send a message parallel and the order is not important. The second one demands to main voter to vote as first, the first voter to vote as second and the last voter to vote as last. The last vote needs to give a voice by Main voter and Voter A (no matter which is first) and then the Voter B may cast a vote. This case shows that some messages must can occur in a specific order and are dependent form the situation when one specific one has been sent earlier.

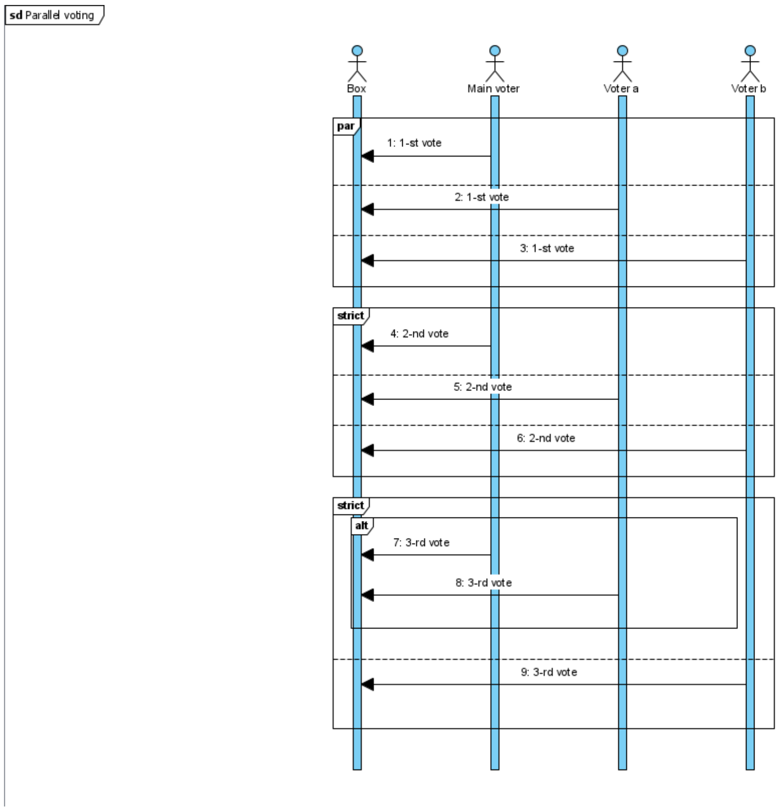


Figure 27: Notation elements for the sequence diagram describing parallel messages [1]

6.2.4 Exercise 1 - Call 112 in case of fire (warm-up)

Please read carefully the task description and create a sequence diagram for the case described below.

There is a protocol for exchanging the information while calling 112 in case of fire and saving the valuable time. Draw a diagram for two actors (a fire reporter and an 112 operator) with a sequence of the messages exchanged between them. The case is that a reporter calls the emergency number and answers the following messages (questions/information) asked/sent by the operator.

- Where is the fire (what floor if apply)?
- What is burning?
- Is human life at stake?
- Fire alarm accepted

6.2.5 Exercise 2 - Drink vending machine.

Please read carefully the task description and create a sequence diagram for the case described below.

Consider a situation when a person orders something from a drink vending machine. In this case we have two actors performing a sequence of exchanging messages (described further). Let‘s name the person ordering a drink as Frank. Frank presses a button to order the chosen drink, thus he sends a message "over the wires" to the machine controller. The machine shows the price to pay, which can be treated as sending the message via "light signal" to the buyer. We assume that a person has enough cash to pay the price. When the coins have been inserted and the buyer has informed machine about that fact by pressing a button (sending a signal), the machine gives a change (if it is needed) or informs the user that it should be, but it is not possible (lack of change money). Frank may accept it (finish transaction without receiving a change) or not. Once all conditions have been met, the machine asks itself to drop the bought beverage down.

6.2.6 Exercise 3 - Self-service cash register.

Please read carefully the task description and create a sequence diagram for the case described below.

Imagine a self-service cash register installed in almost every supermarket. It consists of the following components: a bar code scanner, controller, screen and a scale. All elements contacts with each other by a messages. We are not considering human that is operating this system and assuming that a scanner and a scale are the eyes and ears of the system. The flow is to scan and add a single product to a basket, provided it does exist in the system and the weight of it has been confirmed (to avoid a fraud). When a bar code scanner detects a new product code, it asks the controller (by sending a message with a code value) to check if it does exist in the system. If not, the screen is asked to print the information ("Call the store staff"), otherwise it prints an order to put the product on the scaled basked. After the scale checks the weight, it sends an information to the controller with the result. The controller compares the weight of the product from the scale with a reference value in the system. If the values are very close (let‘s say that equal) a screen is asked to print the information ("scan next product"). Otherwise is asked to print the message: "weigh it again". The maximal possible number of approaches is 3. Once all of them will be used and the weight is still not compliant with expected one the screen is asked to print the message ("Call the store staff").