TDDC88/TDDC93: Software Engineering

Lab 3

UNIFIED MODELING LANGUAGE (UML)

Objectives:

- To gain understanding about use case diagrams and how to implement it
- To gain understanding about class diagrams and how to implement it
- To gain understanding about sequence diagrams and how to implement it
- To understand user requirements in plain text and convert into UML

What is UML?
Unified Modeling Language (UML) is a collection of graphical notations, which are defined using a single meta-model. UML can be used for describing and designing software systems graphically, both at the requirements and design phases of a software life-cycle. The focus is put on an object-oriented style, even if it is not limited to this paradigm.

Ways of using UML
UML can be used in several different ways; with different purposes. The three main categories are:

- *Sketching* - Used by developers and requirement analysts to communicate important aspects of a system. Sketching is rather informal and abstract, leaving out details and focuses on the major parts of a system.
- *Blue prints* - Models the system in a more complete and exact form. Usually only interfaces of subsystems are defined. Using *forward engineering,* the models can sometimes be transformed to source code in a target language. If a tool can read in code and generate models from it, it called *reverse engineering*. If both forward and reverse engineering is supported, the term *round-trip engineering* is being used.
- *UML as a programming language.* UML is used in the context of Model Driven Architecture (MDA), where UML is aimed to be used as a programming language. Using MDA, the engineer creates either Platform Independent Models (PIM) or Platform Specific Models (PSM).

Sketching is the most common way of using UML, even if blue prints and MDA is in the current trend and is getting much attention. In this laboratory exercise we are using UML exclusively for *sketching*.

UML Help
Modeling using UML can often lead to much confusion, especially when it comes to notation issues. See the lectures.

Tasks

In this exercise you will model both the structure and the behavior of a train ticket booking system (from now on abbreviated just *the system*). Imagine a system, such as the Swedish railway's (SJ) booking system. Which are the requirements for such a system? How do you model the most important parts? Which user roles are involved?

We will use UML diagrams for sketching the most relevant aspects of such a system. The diagrams that will be used are:

- Use case diagram - for capturing functional requirements.
- Class diagram - model the most important static structure of parts of the system.
- Sequence diagram - outline the dynamic behavior for one use case.

When you have finished all the tasks you shall report to the lab assistant and give an oral explanation and a demonstration of what you have done and what you have learned from it. Be prepared to answer questions about details in your solution.

Note that modeling is a constructive process which can result in very different diagrams. Hence, there is no single solution to the tasks. The important thing that will be checked by the assistants is if the diagram is understandable and describes the system in a reasonable way.

Get acquainted with a drawing tool

There are many drawing tools on the market. draw.io is an free online tool which is fairly good and follows standard UML quite well. https://www.draw.io/

Task 1. Use case diagram

When starting to model a system from scratch, it is often hard to know where to start. Initially, it is convenient to start with an external view of the system. Hence, the first thing to figure out is which *users* or *roles* that interact with the system. In UML terminology, these are called *actors*, which can be both human beings and other external systems.

In our train ticket system, there can be several different actors. For example, we can define one actor called *Customer*, which is the actor that wants to book and buy a train ticket. Another potential actor is an external *Bank system*, with which the train ticket system communicates, for example, when a customer buys a ticket using a credit card.

To capture the interaction between the train ticket system and the actors, it is convenient to write down *scenarios*. A scenario is basically a plain English text that consists of a sequence of interrelated steps describing the interaction between one or several actors using the system. A *use case* consists of one or several scenarios tied together with a single goal for an actor. The scenarios are written as complete sentences. The first sentence describes how the actor initiates the use case and the last sentence describe how the goal of the use case is fulfilled. It is also explicitly specified which actor that initiates the use case and which other actors that participate in realizing the use case. For example, in the train ticket system one can imagine a use case called *"Buy ticket on-line"*, where both the actors *customer* and *bank system* are involved. Another possible use case could be *"Pickup pre-booked ticket"*. In this use case only one actor is involved, i.e. the customer when he/she picks up the ticket from a machine at the train station[1].

Your task is now to create a use case diagram for the system. Besides the two actors above (customer and bank system) and the two use cases ("Buy ticket on-line" and "Pickup pre-booked ticket"), you shall figure out two more actors and tree more use cases. One of the new use cases shall interact with three of the actors.

[1] Given that the machine is not modeled as an actor itself.

Task 2. Content of the use-cases

UML does not specify how a particular use case should be described and there is no standard way of writing them. The most important aspect of modeling using use cases is actually not the use case diagrams, but simply the use case description text itself. The use case text is always written in natural language and shall be possible to understand by all involved stakeholders, including customers and end-user representatives.

If you have a large use case the text is more readable if you structure the text in a numbered sequence of steps, called the *main success scenario (MSS)* Fowler (2004). In the square below, an example of the content of use case "Buy ticket on-line" is given:

Buy ticket online

Main Success Scenario (MSS):
1. The customer opens a web-browser and loads the start page of the train ticket booking system.
2. The customer selects date, time, and station where he wishes to departure from.
3. The customer selects the destination station.
4. The system shows available seats and price information.
5. The customer selects an alternative and clicks ok.
6. The customer fills in an e-mail address.
7. The customer fills in credit card number and validity date.
8. The system displays the purchase order information.
9. The customer confirms the purchase.
10. The system sends a transaction request to the Bank system.
11. The bank system confirms the transaction.
12. The system displays the booking number and confirmation info in the web- browser.
13. The system sends a confirmation e-mail to the customer.

Extensions:
11a. The bank system fails to perform the transaction
The system displays the failure information.
The customer is asked if it wants to proceed. If yes is selected, return to MSS at step 7.

Besides the MSS we have also written an *extension*. This extension shows the error case of what should happen if the bank system cannot process the transaction. You are

strongly advised to use extensions sparingly, only for very important deviations. Otherwise you will lose readability. Your task is now to write a use case using the style above, for the use case with the three interactions you defined in the last task. Your use case should contain at least two scenarios, i.e. one MSS and one extension. The use case should approximately consist of 10-20 steps. In a written exam about 5 steps is sufficient due to the limited amount of time.

Task 3. Static structure - Class diagram

The use cases helped us to start thinking about the functional requirements of the system from an external point view. In this exercises we shall instead design parts of the static structure of the system using a class diagram.

The first thing to do is to identify potential classes. This is not always easy to do at once, so several iterations need to be done during the class design. One possible way of identifying potential classes is to do a so called *noun analysis*, i.e. to mark all words in a use case which are nouns. A noun is a word that is used to name a person, an animal, a place, a thing or an abstract idea. For example, in the sentence "The customer enters a pin-code", the underlined words are nouns. The marked nouns can then be used as "potential classes" for a class diagram. However, make sure that only relevant nouns are used. To become a class the noun shall represent a true or abstract object being manipulated by the system. Sometimes it is more natural to make a noun becoming an attribute, for instance that the pin-code is an attribute of the class customer. You also need operations, that is, functions you can call to make something happen. For instance, you can have an operation of a customer class called changePinCode()

Your task is now to create a UML class diagram for the train ticket system, by first doing a noun analysis of use case "Buy tickets online". Besides the use case, the class diagram should model the following aspects:

- The system holds information about a fixed set of *train journeys*, i.e. a path between stations where the train just makes short stops. E.g. if a train starts in Malmö, makes a stop in Linköping and has Stockholm as final destination, the train journey is Malmö-Stockholm.
- The model should describe how train journeys, stations, customers, and bookings are related.
- The customer given in the use case has no member card and makes only a single time purchase of a ticket. Add another customer category, which has a year card subscription, i.e. who has a login account and pays a one year flat rate.
- Model only the parts that can be seen as parts of the system, e.g. do not include the bank system in your class diagram.
- The model should typically include between 5 and 10 classes with 1-3 attributes and operations each.

Some tips:

- Use only simple class diagram notations, such as generalization, attributes, associations, and multiplicity.
- Try to focus on the most important aspects of the system. Avoid getting into details. You don't need to model seats in the trains and you don't need a perfectly constrained model that rules out all unintended instantiations.
- Try to be pragmatic and create a model that you find reasonable.
- Some nouns do not need to be described as classes and associations, but just as attributes to other classes. E.g. a booking number is important to include in the diagram, but do not need to have a class of its own.

Task 4. Dynamic behavior - Sequence diagram

A UML sequence diagram is a form of interaction diagram that shows how a collection of objects interact with each other. Instead of showing the static structure as in class diagrams, a sequence diagram describes a certain behavior of the system. A sequence diagram captures typically the behavior of a scenario.

In this task, you shall model a sequence diagram describing the main success scenario of the "buy ticket online" use case. In contrast to the class diagram, you shall now include external actors, such as, the customer and the bank system. Furthermore, you shall include some of the internal classes of the system that you defined in task 3.

The Violet UML editor has some expressiveness limitations when it comes to sequence diagrams. The draw.io tool is much better, but it requires some planning of the layout of the diagram to avoid lot of adjustments. Therefore, this exercise should not be done using your computer, but with ordinary *pen and paper*. UML is actually very useful when sketching with pen and paper, or during a meeting on a whiteboard.

If there are alternate branches of a flow of messages, you use an alt-fragment, or an opt-fragment with guards to represent what happens if the tickets are sold out or the bank account is empty.

Some tips:

- Use only simple notations, such as synchronous messages, lifelines with activations, return messages, and fragments. (see lectures).
- A diagram with 5-7 lifelines is reasonable.
- Sketch the diagram first on one paper; then draw it more nicely on another piece of paper when you feel that you have a solution.
- Remember that UML sketching is about showing the most important part of the system, i.e. we should avoid unnecessary details.
- If you are unsure of the UML notation, please review the lecture notes about sequence diagrams (see lectures).

Examination

When you are done with the tasks please demonstrate them for your lab assistant during a lab session. He/she will ask questions and you should be able to explain and motivate your design.

Reference

Martin Fowler, *"UML Distilled Third Edition - A brief guide to the standard object modeling language"*, ISBN 0-321-19368-7, Pearson Education Inc, Boston, USA, 2004

http://www.uml-diagrams.org/ currently maintained by Kirill Fakhroutdinov

Object Management Group (OMG)*, Unified Modeling Language (UML) specification, version 2.1.1*, available at: http://www.omg.org/technology/documents/formal/uml.htm

Other resources

Microsoft Visio

The Violet UML tool has some limitation. A more advanced alternative is to make use of Microsoft Visio. A very good stencil and template library for UML 2.0 for MS Visio can be downloaded from: http://www.softwarestencils.com/uml/

A free web-based drawing tool can be found at:
https://www.draw.io/