INFO 101 – Introduction to Computing and Security / Tutorial – week 6

Prof. Dr. Rui Abreu and Prof. Dr. Franz Wotawa
IST, University of Lisbon, Portugal and Technical University of Graz, Austria
rui@computer.org, wotawa@ist.tugraz.at

November 19, 2020

1 Introduction

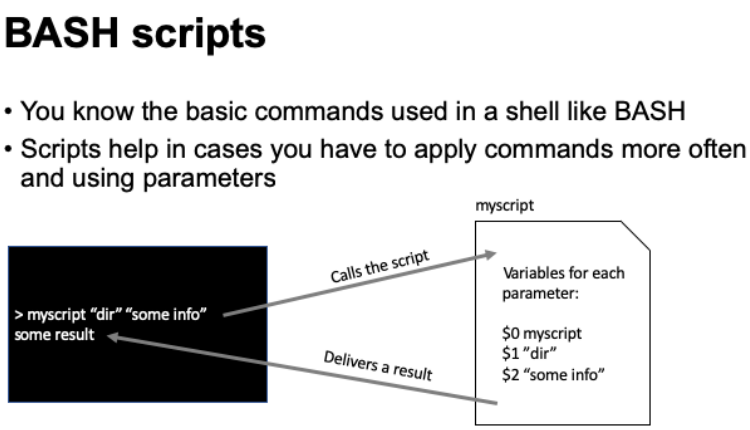
This time the tutorial has two objectives, i.e.:

- Show how to program shell scripts.
- Discussing the use of UML class diagrams, use case diagrams, and state chart diagrams again.

2 Shell scripts

We already discussed shell commands and shell scripts a lot in previous lab sessions. However, it seems to be not clear how parameters can be passed to a shell script. In addition, it is not clear how to give shell scripts a name. For the latter, a name should reflect the content of a script. We also may add an extension like `.sh` or `.bash` to a script. For example, if we want to make a script that delivers for us all files having a certain extension, e.g., `.docx`, we might call it `countFiles` but any name is sufficient. Sometimes the name is provided in an example. For temporary files, i.e., files that are used to store content during an execution of a shell script or commands in the shell, but not of use afterwards, we also should define a name. This name should be not used in other files. Hence, something like `.temp`, or `.dummy` is usually a good choice.

Regarding parameter passing have a look at the following figure:



In this figure, we assume a script named `myscript` to be called (see left part of the figure). There are two parameters `dir`, `some info`. Note that we have to use quotes whenever we want to use

a string as one parameter comprising spaces. E.g. `some info` has to be passed as `"some info"` wheres `dir` can be passed as `dir` or `"dir"`. When a script is executed, the actual parameters (i.e., the given parameters) are passed to the script. On side of the script the parameters are stored in variables `1`, `2`, ..., `9`, and the name of the script (including its path) in variable `0`. The value of the variables can be obtained using `$` before the name, i.e., `$0`, `$1`, ..., `$9`. This is depicted at the right side of the figure.

We can easily see how this binding is working. For this purpose, we write the following shell script called `paramtest`:

```
#!/bin/bash

echo "This is script ${0}"
echo $1
echo $2
echo $3
echo $4
echo $5
```

Note that in `echo "This is script ${0}"` we reference variable `0`, i.e., the name of the script. Because this variable reference is used in a string, we have to use `{}`. This is a special case in shell scripting.

What is this shell doing? First, it prints the name of the script and afterwards the value of 5 parameters. Let us test the script:

```
ruimaranhao@greyhound:labs ruimaranhao$ paramtest par1 "par2" par3
This is script ./paramtest
par1
par2
par3


ruimaranhao@greyhound:labs ruimaranhao$
```

In this case we provided 3 arguments/parameters. The script returns it name and the value of 5 parameters causing two empty lines before the next prompt of the shell. Let us try another example:

```
ruimaranhao@greyhound:labs ruimaranhao$ paramtest 1 2 "this is the third parameter" 4 5 "and the
sixth one" 7
This is script ./paramtest
1
2
this is the third parameter
4
5
ruimaranhao@greyhound:labs ruimaranhao$
```

In this case, the 5 parameters are printed but not the 6th and 7th, which is the correct behavior of our script. For more information, please have a look at the documentation provided in previous lab exercises as well and carry out experiments using shells for both the command line and the scripts. Please also use the manual pages that can be accessed using the `man` command with the name of the shell command you want to know more about. For example, `man man` returns you information of the `man` command itself.

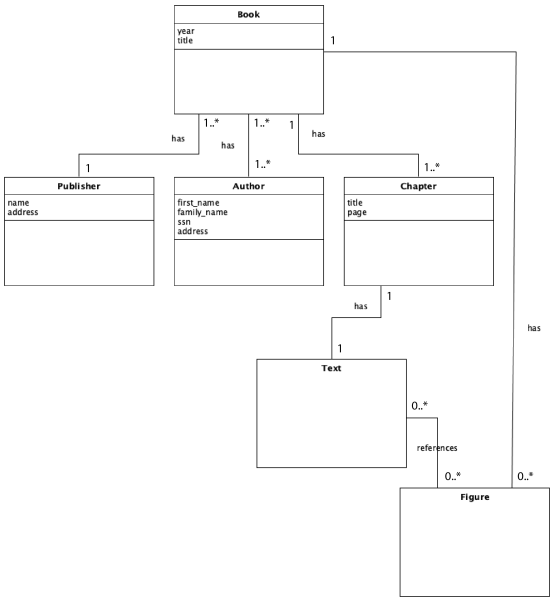
3 UML diagrams

3.1 Class diagrams

Let us consider the following simple example summarizing the parts of a book. *A book comprises chapters each of them having a title and a text, and starting at a certain page. A book may also have figures that might be referenced from different places in the text. A book also has at least one author, a publisher, the year of publishing, and a title. We may also use additional information about an author or publisher.*

Note that such examples can always become more and more complicated when you think about what information you want to store and what tasks you want to capture. In order to come up with a class diagram we have to identify the main concepts, i.e., classes. Remember the noun method where we first focus on nouns in a textual descriptions. Nouns describing basic concepts or physical objects are most likely to become classes. Some concepts like dates, numbers, or strings might be considered classes but are more often used directly in attributes of classes (providing that there is we only need the concept once for the class where we want to add the attribute).

From our textual descriptions of books, we easily obtain the following classes: book, chapter, title, text, page, figure, author, publisher, year, and book title. Titles are usually like years or pages, only basic types and can be easily represented as attributes. When considering the relations that are also indicated in the text using verbs like 'comprises', 'has', 'are in', etc. we are able to come up with a class diagram such as follows:



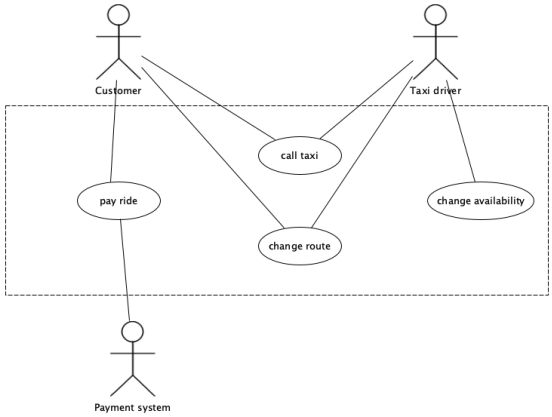
Note that we also added some attributes not mentioned in the text. For example, every book has to have a title (and not only a publishing year). A publisher and an author also have names, addresses and other information. In practice, such UML diagrams are used to discuss the basic concepts that should be implemented in a program. They are used as basis for discussions and most likely evolve over time. More and more information is added until we know the application area sufficiently well to implement the program.

3.2 Use case diagrams

Let us develop a use case diagram for a taxi app! This app should allow people calling a taxi for a specific time and place, and maybe also for a certain destination. We should be able to pay the taxi with the app and also to reroute the taxi if necessary. A taxi driver can access the app to offer a ride for a given time, i.e., stating his or her availability.

For such examples, it is first important to identify the users (actors) of the app. In our case this will be obviously the customer and the taxi driver. In addition, for payment there might be an electronic payment system. The purpose of identifying users is also to clarify the boundaries of the system. Afterwards, we have to identify the use cases (or functions the system needs to offer). From the given text we see that we have to *call a taxi*, *pay the taxi*, *reroute the taxi*, *change availability*. These 4 functions have to be represented in the UML use case diagram. We also have to identify the actors involved in a function. When calling a taxi we need the customer and a taxi driver, For rerouting we also require both actors. For changing availability only the taxi driver is responsible. For paying the fare, we need the customer and the payment service.

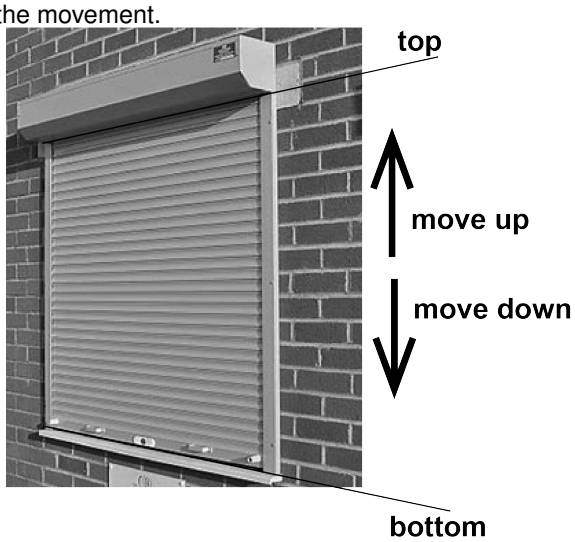
Considering all these finally leads to the following diagram:



Note that in practice we would also describe the functions (i.e., use cases) in more detail in a structured textual form. We would state what is need for a use case and what we want to achieve. For the practical part (and not if otherwise stated) the diagram is sufficient.

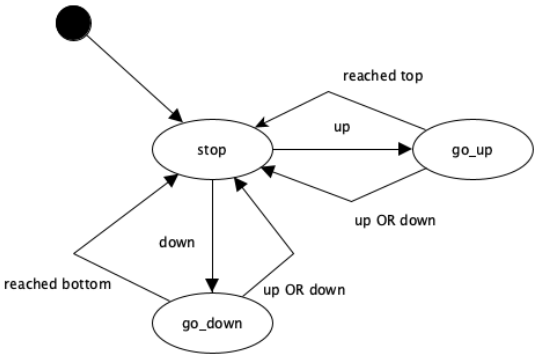
3.3 State chart diagrams

Draw a UML state chart diagram that captures the behavior of a simple roller shutter system, where we have two buttons. One button is used to move the roller shutter up and the other is to move it down. Whenever the roller shutter is moved (either up or down), we can stop it when pressing a button. Afterwards, we can again continue going up and down using the two buttons. The roller shutter automatically stops when reaching the top or bottom of the window depending on the direction of the movement.



In order to come up with a state chart, we need to identify the states first. Informally speaking, a

state of a system characterizes a special thing/behavior of a system that should be distinguished. For example, a car can be stopping or moving. When it is moving we can do different things, e.g., going from one place to another. When stopping we cannot do that. In our example, the roller shutter can stop, move up or move down. Hence, we have at least three states, e.g., *stop*, *go_up*, *go_down*. We now have to think about how to come from one state to another. From stopping to going up we need to press the up button. Hence, we have an action 'pressing the up button', which can be represented by *up*. Similarly, we have an action *down*. Moreover, in the last sentence of the example, it is mentioned that we might reach the bottom or top. Again, we can interpret this as actions *reached_bottom* and *reached_top* respectively. When bringing all these together we finally obtain:



Note that we may come up with other solutions as well. We might distinguish more states like being also at the top or bottom. In practice, there are consequences arising from considering states that reflect a determine a certain behavior. However, for the practical part it is sufficient to come up with a solution that matches a given textual description.