GABRIELE

General Agent Based
Repast Implemented Extensible
Laboratory for Economics

Users Manual

Gianfranco Giulioni

April 6, 2018

To Gabriele

Contents

List of variables

. 27

$D_{f \leftarrow f,j}$ Demand of goods for investment pupose. 27

. 27

. 27

List of Parameters

Part I

Setting up GABRIELE

In this part of the manual we describe two ways of setting up for running simulations. The first one is the standard Repast procedure, where one can take advantage of various wizards to set up and run simulations. Although this provides several facilities, some user could feel uncomfortable with this standard process and would like to revert to a more basic procedure. The streamlined installation is for these users. It behooves to point out that the streamlined procedure was especially designed for running the model in the local machine and in batch mode. Therefore, the supporters of the Unix like command line interface will appreciate it. To help clarifying, it is worth to say that the streamlined procedure was developed to avoid the slowness of the wizards when continuously checking the effects of introducing new lines of code. It allows to run the model from the command line by typing two keys (i.e arrow up key - to recall the execution command, and the enter key).

Chapter 1

Standard Installation

1.1 Installation

In this section we describe the standard installation process needed to prepare the system to run simulations. After taking the steps described below, the user should be able to run the model regardless of the operative system s/he is using.

The model needs Repast Symphony (RS), who in turn needs the Java Development Kit (JDK). Therefore we need first to check if the JDK is installed in the system and install it if needed. Once JDK is properly running, we have to install RS. Finally the model can be installed and run in RS.

1.1.1 Java Development Kit (JDK)

Check the list of installed software to know if JDK is installed in your system. If yes, note the JDK version. Alternatively, you can open the command line interface of your system, type `javac -version` and hit the return key.

Once verified if JDK is installed and, if yes, its version, visit the following URL:
http://www.oracle.com/technetwork/java/javase/downloads/index.html
to know which is latest released version of JDK.

If JDK is not installed in your system or if it is not at the latest release, follow the instruction found in the JDK download page to install or upgrade it. You can also search the internet for alternative ways to install or upgrade the JDK on your system.

This installation phase is complete when the `javac -version` command returns what you expect. If not, you should fist check if the folder containing the JDK executables are in your execution path and add it manually if needed. Furthermore, some Linux distribution must be informed on which JDK to use using the `update-alternatives` command.

1.1.2 Repast Simphony (RS)

The Repast suite website: http://repast.sourceforge.net has all the information needed to download and install RS.

Note that RS is provided as a plugin of the eclipse Integrated Development Environment. The Repast development team provides a customized version of eclipse, so you could encounter problems with already installed versions of eclipse. Note that the streamlined installation provided below show how to avoid using eclipse.

1.1.3 GABRIELE

GABRIELE has to be installed as an eclipse RS project. We will give hereafter the instructions to achieve this goal.

First of all, open eclipse.

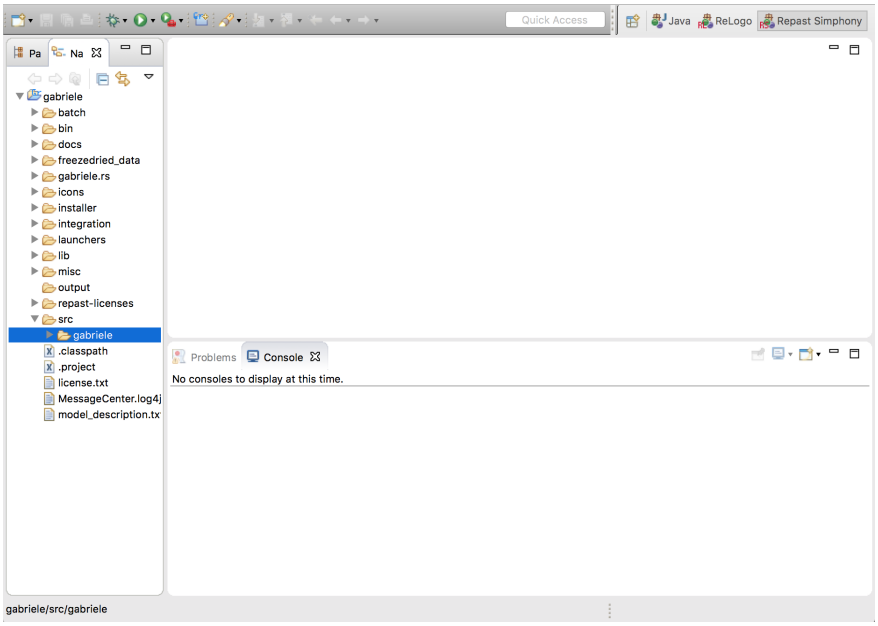
Suppose your workspace has the following path:
`/Users/coolcoder/Documents/workspace`

Open the RS perspective (window → open perspective).

Create a new RS project called `gabriele` (file → new → Repast Simphony project)

This creates the `gabriele` folder and a series of sub folders inside the workspace:

The following figure show the `gabriele` project folders tree.



Now, the `gabriele` files have to be added to the just created RS project folders tree.

We give here two alternatives: via git and using a zipped archive.

Using git

As you probably know, this is a popular way to share code. To fetch GABRIELE code, a git client need to be installed in your system. Many system comes with a git client already installed; if it is not your case, you have to install it. Mac and windows users can consider to install the GitHub Desktop software.

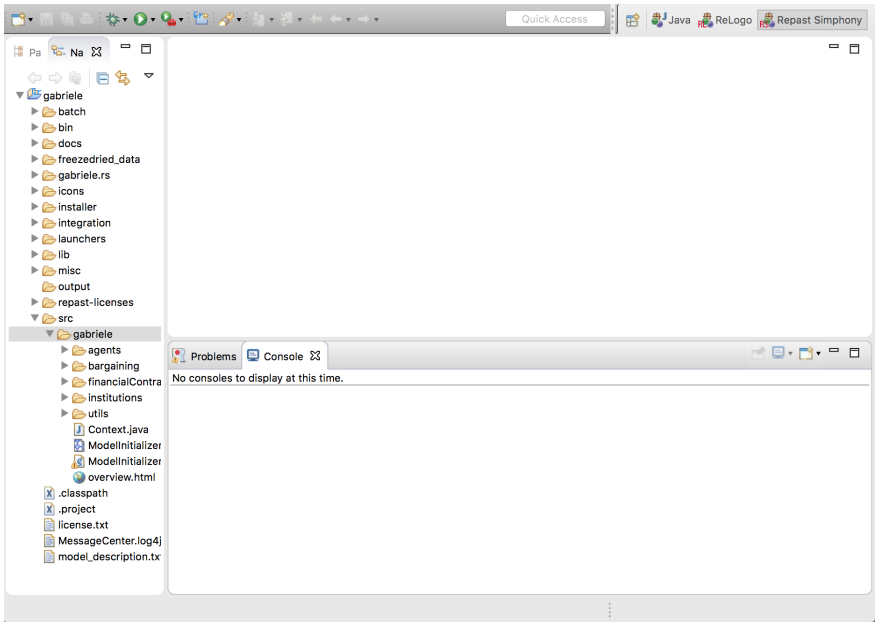
You can verify if git is installed in your system by checking if your command line interface recognize the `git` command. If your check is successful, change directory to the gabriele project folder:
`cd /Users/coolcoder/Documents/workspace/gabriele`
and type the following commands:

```
git init
git remote add origin https://github.com/ggiulion/gabriele.git
git fetch origin master
git reset --hard FETCH_HEAD
```

Then if you plan to make your code changes available on GitHub, add the command:
`git push --set-upstream origin master`

Now, the gabriele files should show up in the RS project folders. Refresh the gabriele RS project with the navigation tab selected in the side bar (file → refresh) to make them visible in eclipse.

The following figure show how the `src` sub folder should look like.



Using a zip archive

Point your browser to
`https://github.com/ggiulion/gabriele`
Click the "clone or download" button and choose "download zip".

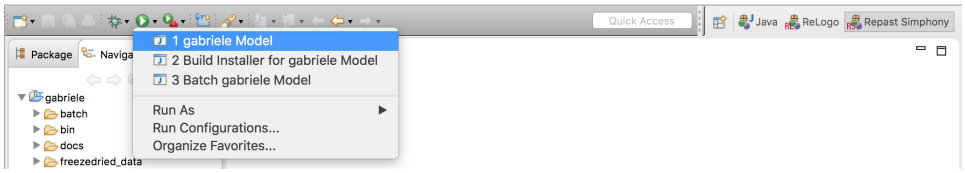
This will download the `gabriele-master.zip` file in your system.

Unpacking it creates the gabriele-master folder. Move the whole content of this folder in the gabriele RS project folder:
`/Users/coolcoder/Documents/workspace/gabriele/`
Choose to overwrite existing files and folders if you will be asked (this will merge folders). Now refresh eclipse (file → refresh).

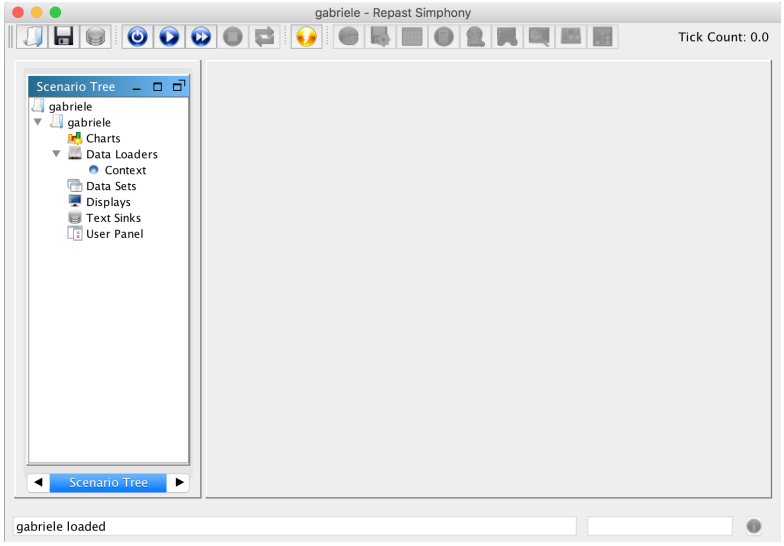
1.2 Testing the Installation

First you have to start the RS GUI window. To do so, click on the down black arrow highlighted by the red circle in the following picture

After clicking, a menu opens as shown by the following figure. Click the `gabriele Model` item



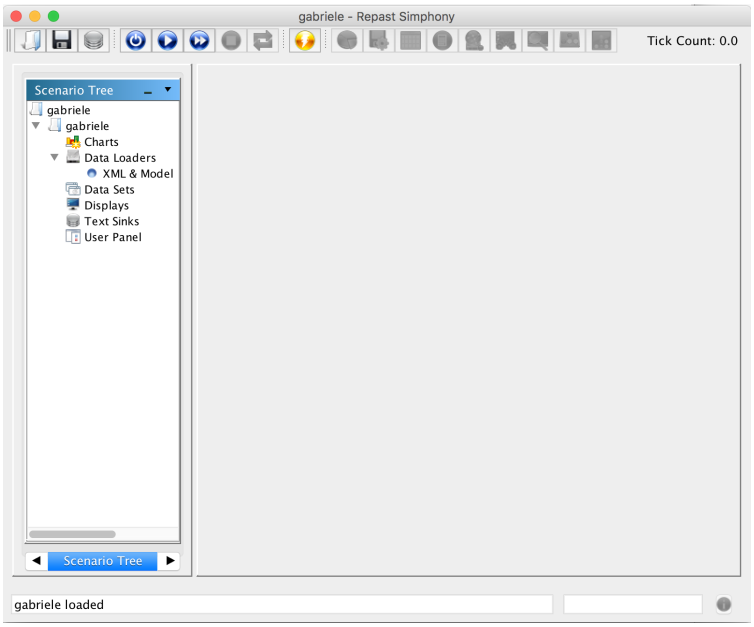
After a while, the RS GUI (displayed in the following figure) will show up



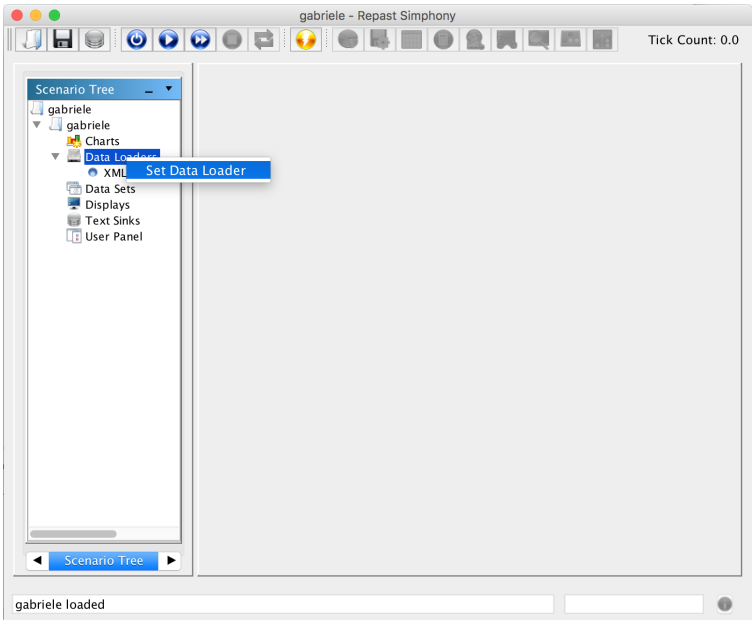
Check the `Data Loaders` item in the Scenario Tree. If the `Context` sub-item is reported (as in the last reported screen shot), the model can be run. Other ways, the additional settings described in next section are needed.

1.2.1 Setup the data loader

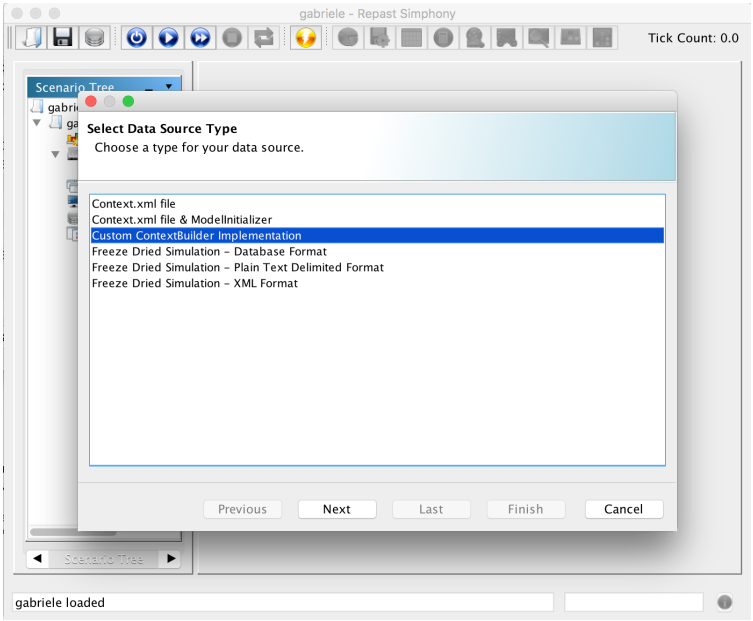
If `Data Loaders` reports the `XML & Model` (as in next screenshot)



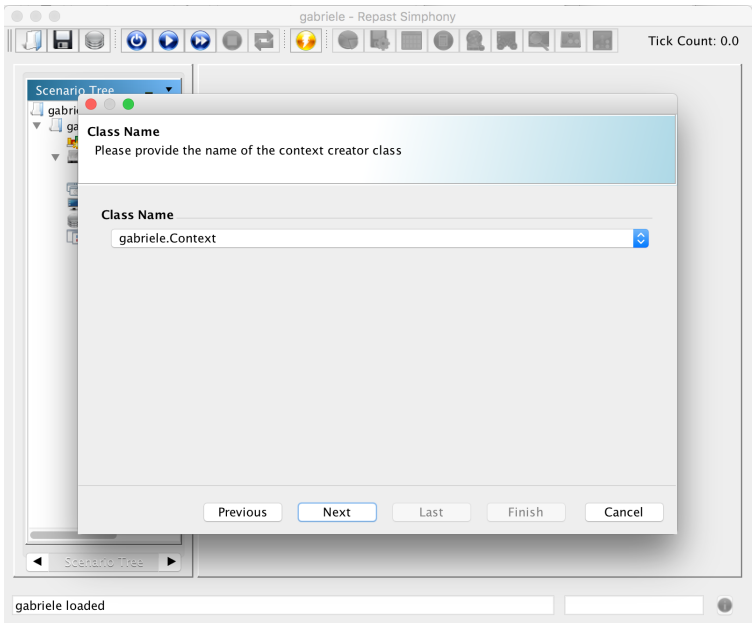
you have to setup the custom dataloader. As you can see in the scenario tree under the Data Loaders item, the XML & Model data loader is present. This must be changed in the custom loader for this model. To do so right click on the Data loaders item and choose set Data Loader as shown in the following figures



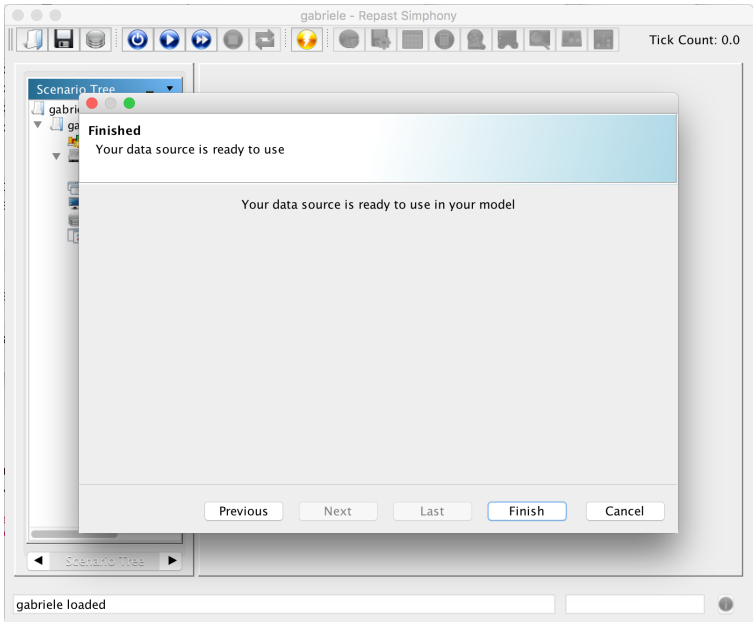
The following new window will appear



Choose Custom ContextBuilder implementation and click next: A new window with a proposal will appear.

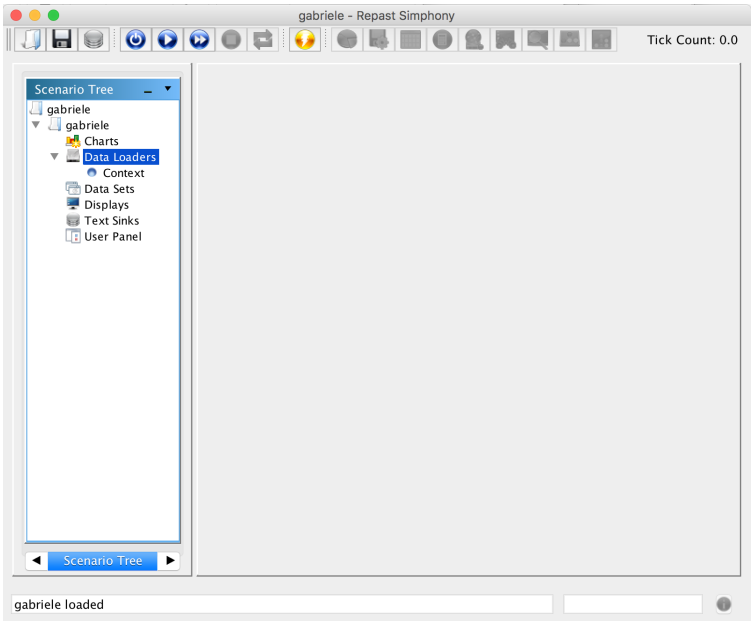


Clicking next will change the window as follows



Click finish.

Now, the previous data loader is replaced by the GABRIELE data loader (Context) as shown by the following figure



Click on the floppy disk icon to make the change permanent.

1.2.2 Running GABRIELE

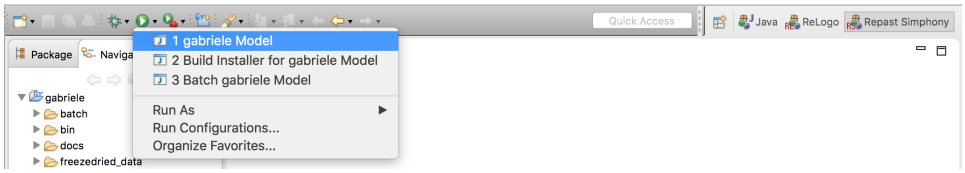
There are two ways of running RS models: The GUI and the BATCH mode.

The GUI mode can be of great visual impact because several monitoring devices continuously updating during the run can be added to the RS GUI window. The flip side of the coin is that these devices slow down simulation execution. Second, the simulation runs exclusively in a machine running an X server. Notwithstanding the GUI mode can be a valid tool during the model development. When massive simulations are performed, the BATCH mode should be used instead. BATCH mode runs are faster because all the graphics elements are turned off. Furthermore, the absence of graphics makes it possible to run the model in parallel on several machines. It is worth saying that RS has very useful facilities for running a model in parallel.

Let us start this section from the very beginning i.e. with the eclipse software just opened and show how to run the model both in GUI and BATCH mode.

First of all, click on the down black arrow highlighted by the red circle in the following picture

After clicking, a menu opens as shown by the following figure.



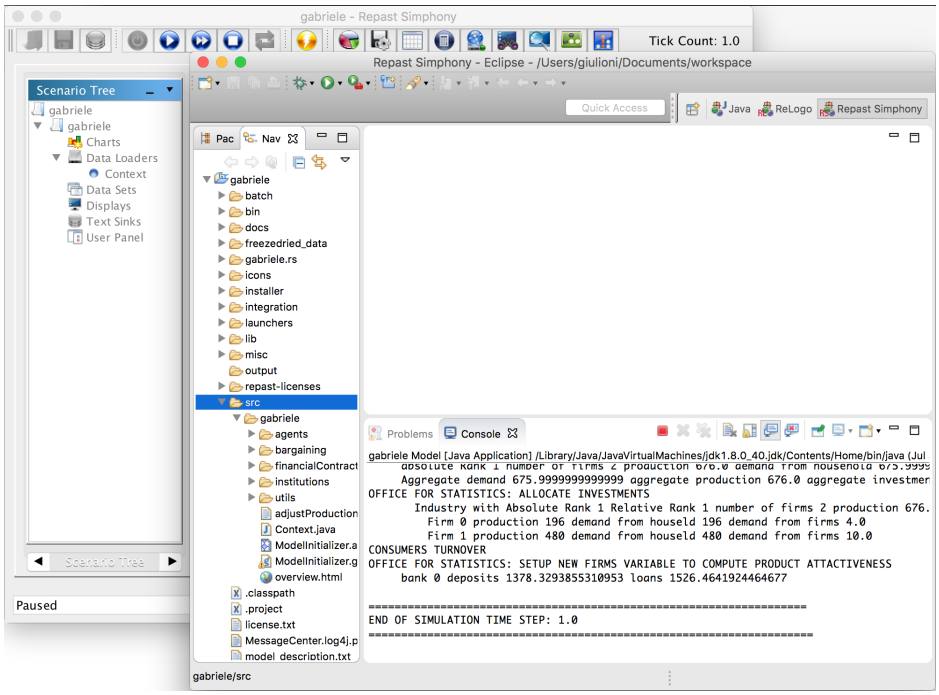
The GUI execution is activated by clicking the `gabriele Model` item, while the BATCH execution can be started by clicking the `Batch gabriele model`.

Running in GUI mode

RS GUI window opens by clicking the `gabriele Model` item. Check if Context is listed under the Data Loaders item of the scenario tree.

Use the RS GUI window intuitive buttons to interact with the simulation. A more detailed description on how to control the simulation is given at page 24 “Repast Java Getting Started” document available in RS web site.

The current version of the model has no runtime monitoring graphical element, so one can check the progress of the simulation watching at the tick count number in upper right corner of the RS GUI window or looking at the text output in the console panel of Eclipse window as shown in the following figure.



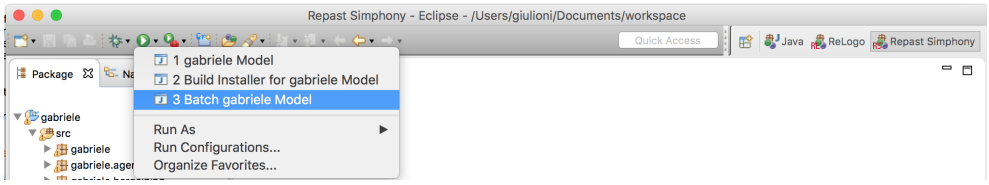
On the other hand, GABRIELE record data in text and csv files that will be created in the model base directory:

`/Users/coolcoder/Documents/workspace/gabriele`
Data files can be easily identified because their name starts with `zdata`.

The file `zdata_aaa_readme.txt` gives a description of the contents of all the data files.

Running in BATCH mode

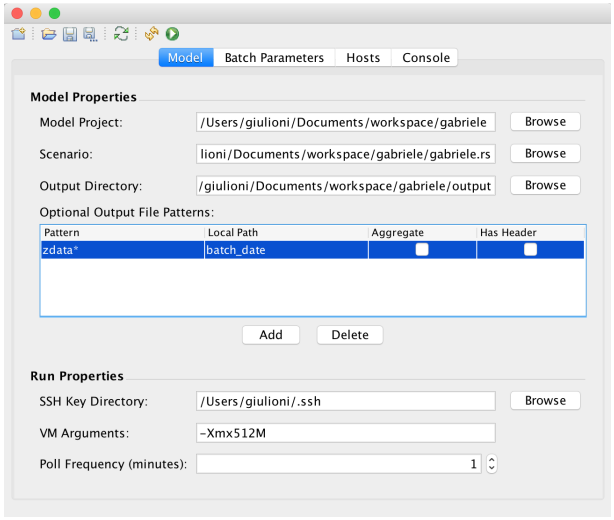
Clicking on the `Batch gabriele model` as in the following figure



activates the batch run configuration wizard.

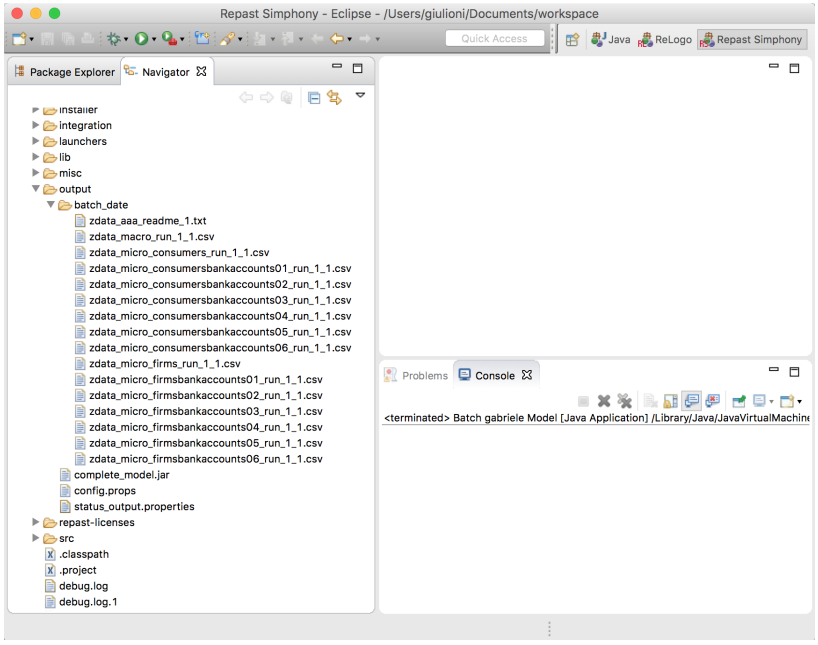
We point the reader to the "Repast Simphony Batch Runs Getting Started" document available in RS website for a full description of the wizard.

We only point out that the present version of the model uses custom output recording techniques. Therefore, the RS File Sink are not used. This implies that the "Optional Output File Patterns" must be set as in the following figure



This configuration instructs RS to fetch all the file whose name begins with `zdata` from all the machines running the model in parallel and move them in an output sub-folder named `batch_date`. Needless to say that the sub-folder name can be chosen at your convenience, and it can be changed at each batch run in order to avoid overwriting the data (perhaps you want to have memory of the time of your runs. It is why we add `_date` to the folder name).

The following figure shows the contents of the model output folder after a local batch run using Eclipse navigator panel


Repast Simphony - Eclipse - /Users/giulioni/Documents/workspace
Quick Access
Java
ReLogo
Repast Simphony
Package Explorer
Navigator
installer
integration
launchers
lib
misc
output
batch_date
zdata_aaa_readme_1.txt
zdata_macro_run_1_1.csv
zdata_micro_consumers_run_1_1.csv
zdata_micro_consumersbankaccounts01_run_1_1.csv
zdata_micro_consumersbankaccounts02_run_1_1.csv
zdata_micro_consumersbankaccounts03_run_1_1.csv
zdata_micro_consumersbankaccounts04_run_1_1.csv
zdata_micro_consumersbankaccounts05_run_1_1.csv
zdata_micro_consumersbankaccounts06_run_1_1.csv
zdata_micro_firms_run_1_1.csv
zdata_micro_firmsbankaccounts01_run_1_1.csv
zdata_micro_firmsbankaccounts02_run_1_1.csv
zdata_micro_firmsbankaccounts03_run_1_1.csv
zdata_micro_firmsbankaccounts04_run_1_1.csv
zdata_micro_firmsbankaccounts05_run_1_1.csv
zdata_micro_firmsbankaccounts06_run_1_1.csv
complete_model.jar
config.props
status_output.properties
repast-licenses
src
.classpath
.project
debug.log
debug.log.1
Problems
Console
<terminated> Batch gabriele Model [Java Application] /Library/Java/JavaVirtualMachine

Chapter 2

Streamlined Installation

2.1 Installation

This chapter reports the instructions for installing and running the model in Unix like Operating Systems using a command line approach. Therefore, the instructions will be also valid for Linux and recent Mac machines.

The examples and the command line outcomes given below relate to a user named `coolcoder`. You should easily be able to adapt the paths to you own user account.

The following colors are used:
red to denote a command;
blue to denote an ordinary file in command line output;
green to denote an executable file in command line output;
magenta to denote the contents of text files.

2.1.1 Java Development Kit (JDK)

Follow the instructions given in the previous chapter to install or update JDK if needed.

2.1.2 Repast Simphony (RS)

Also in this case, you can follow the instructions given in the previous chapter. However, the process therein described implies the installation of eclipse. We will give here an alternative way to install the RS library and to use it directly.

First of all you have to download all the RS library packages.[1]

You can download all the jars by using the `wget` command with the recursion option (`-r`).[2]

[1] Note that the installation process described in RS web site implies downloading the RS library and put them in the eclipse plugins folder.

[2] If the command is not available in your system you have to install it.

The following steps have to be taken to install RS.

Suppose, for example, you have the directory `/Users/coolcoder/abm_java_libraries`. Create the directory

```
mkdir repast
```

and move into it

```
cd repast
```

Download the files from the RS repository

```
wget -r -l1 --no-parent -nd --no-check-certificate
            https://repo.anl-external.org/repos/repast/plugins/
```

Some minutes are needed to complete the download.

The directory now should contain many jar files.

Give the following command:

```
ls *.jar|awk -F'.jar' '{print "unzip "$0" -d "$1}'|sh
```

Each jar file now has the corresponding folder. Remove all the jar files by typing:

```
rm *.jar
```

Now the RS library is installed in your system and is ready to be used. To test your installation type the command:

```
java -cp /Users/coolcoder/abm_java_libraries/
              repast/repast.simphony.runtime_<version>/lib/*:
         /Users/coolcoder/abm_java_libraries/
              repast/repast.simphony.runtime_<version>/bin
         repast.simphony.runtime.RepastMain
```

where you have to replace `<version>` with the version identification number (for example `2.3.0`).

After a while, the RS GUI window should pop up.

2.1.3 GABRIELE

Now you have to choose or create the GABRIELE destination folder. Suppose it is called `models` and has the following absolute path:

```
/Users/coolcoder/models
```

You can use again the two methods described in the previous chapter (git or compressed archive) to install the model.

Briefly, using git, change directory in `models` and give the following command:

```
~/models$ git clone https://github.com/ggiulion/gabriele.git
```

That's it!

Otherwise, you have to download the zip archive: point your browser to

```
https://github.com/ggiulion/gabriele
```

Click on the "clone or download" button and choose "download zip".

This will download the `gabriele-master.zip` file in your system.
Move the archive in
`/Users/coolcoder/models`
Unpacking it, the `gabriele-master` folder is created.
Rename the `gabriele-master` in `gabriele`.
Delete the `gabriele-master.zip` file.

Regardless of the method used, you should have the following directories tree:

```
/Users/coolcoder/models/gabriele
/Users/coolcoder/models/gabriele/src
/Users/coolcoder/models/gabriele/docs
/Users/coolcoder/models/gabriele/gabriele.rs
/Users/coolcoder/models/gabriele/scenario
```

Now, `cd` into the `gabriele` directory and get its absolute path

```
~/models$ cd gabriele
~/models/gabriele$ pwd
/Users/coolcoder/models/gabriele
```

Save this information because it will be used in the configuration phase.
We will refer to it as the model base directory.

2.2 Testing the installation

2.2.1 Configuration

Create a new directory outside the model base directory.
We will refer to it as the data directory.
Suppose the data directory is called `gabriele_data` and has the following absolute path:
`/Users/coolcoder/Documents/gabriele_data`
`cd` into the data directory.
Find out the Repast installation directory:

```
~/Documents/gabriele_data$ sudo find / -name "repast.simphony.core*"
Password:
/Users/coolcoder/abm_java_libraries/repast/repast.simphony.core_2.3.1
```

In this expression, `/Users/coolcoder/abm_java_libraries/repast` is repast base directory and `2.3.1` is repast version.
Prepare a text file named `paths.txt` having the repast base directory in its first line, repast version in the second line and the model base directory in the third line. The file content should look as follows:

```
/Users/coolcoder/abm_java_libraries/repast
2.3.1
/Users/coolcoder/models/modelJasss
```

You must adapt the paths and the repast version of this file to your settings. Save this file into the data directory.

Move the `configure` file from the gabriele scenario folder to the data directory:

```
mv /Users/coolcoder/models/gabriele/scenario/configure .
```

The contents of your data folder is now:

```
~/Documents/gabriele_data$ ls
configure
paths.txt
```

Make the `configure` file executable and run it:

```
~/Documents/gabriele_data$ chmod +x configure
~/Documents/gabriele_data$ ./configure
```

This creates three additional files:

```
~/Documents/gabriele_data$ ls
compile
configure
paths.txt
run_batch
sourcefilespath
```

Make the `compile` and `run_batch` files executable:

```
~/Documents/gabriele_data$ chmod +x compile
~/Documents/gabriele_data$ chmod +x run_batch
```

2.2.2 Running GABRIELE

We recall that the streamlined installation was built to run the model in BATCH mode in a fast way avoiding the slowness of the batch wizard. Therefore we will only give instruction for the command line batch run.

First of all compile the model by typing:

```
~/Documents/gabriele_data$ ./compile
```

The batch run is started with the following command:

```
~/Documents/gabriele_data$ ./run_batch
```

When the run completes, you will find the files containing the output of your simulation inside the data directory. The data file have the `zdata_` prefix for ease of their identification:

```
~/Documents/gabriele_data$ ls
compile
```

```
configure
paths.txt
run_batch
sourcefilespath
zdata_aaa_readme.txt
zdata_macro_run_1.csv
zdata_micro_consumers_run_1.csv
zdata_micro_consumersbankaccounts01_run_1.csv
zdata_micro_consumersbankaccounts02_run_1.csv
zdata_micro_consumersbankaccounts03_run_1.csv
zdata_micro_consumersbankaccounts04_run_1.csv
zdata_micro_consumersbankaccounts05_run_1.csv
zdata_micro_consumersbankaccounts06_run_1.csv
zdata_micro_firms_run_1.csv
zdata_micro_firmsbankaccounts01_run_1.csv
zdata_micro_firmsbankaccounts02_run_1.csv
zdata_micro_firmsbankaccounts03_run_1.csv
zdata_micro_firmsbankaccounts04_run_1.csv
zdata_micro_firmsbankaccounts05_run_1.csv
zdata_micro_firmsbankaccounts06_run_1.csv
```

Part II

Understanding GABRIELE

This model aims at reproducing the dynamics of an economic systems.

The functioning of a dynamic system is described by giving information on the following elements:

- the components of the systems,
- how they behave and
- how they interact with each other.

Chapter 3 aims at showing the **model components**.

Chapter 4 describes the dynamics of events. This includes the initialization phase which is performed just once at the beginning of the simulation and the **sequence of events** repeated in each time step.

Chapter 3

The components of the system

Figure 3.1 gives a visual representation of the system components.

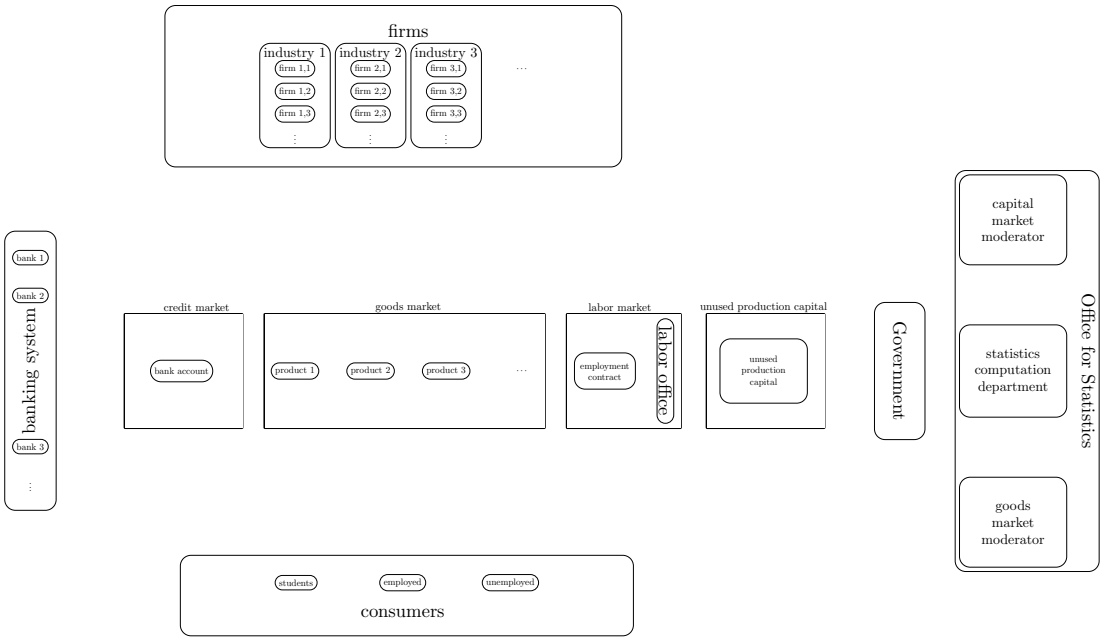


Figure 3.1: Components of the economic system

For a systematic description of the system components, we divide them in the following categories:

- agents;

- goods;
- financial assets.

3.1 Agents

Let us distinguish between multi-instances and single-instance agents.

In this version of the model, Multi-instances agents are

- consumers
- firms
- banks

while single-instance agents are:

- government
- labor office
- Office for statistics

Multi-instances agents are grouped in sectors: the consumers, firms and banking sectors. Adding the government to these three sectors, we obtain the usual categorization of the economy into the four institutional sectors: households, firms, financial sector and public sector. Figure 3.2 highlights the institutional sectors in light blue. The other two single-instance agents are highlighted in gray: the Labor Office operates in the labor market, while the Office for Statistics, has a global view of the system. The latter computes aggregate variables making them available to all the other agents. Furthermore, it uses this information to moderate the goods and production capital markets.

Figure 3.2 also shows the internal structure of the firms and consumers sectors.

Consumers are classified in students and non-students. The latter in turn can be employed and unemployed. Therefore, each consumer of the model falls into one of the these three groups: student, worker and unemployed.

The internal structure of the firms sector is designed to allow for product differentiation. In case the researcher is interested in having multiple products, firms are organized into industries that groups firms making similar item.

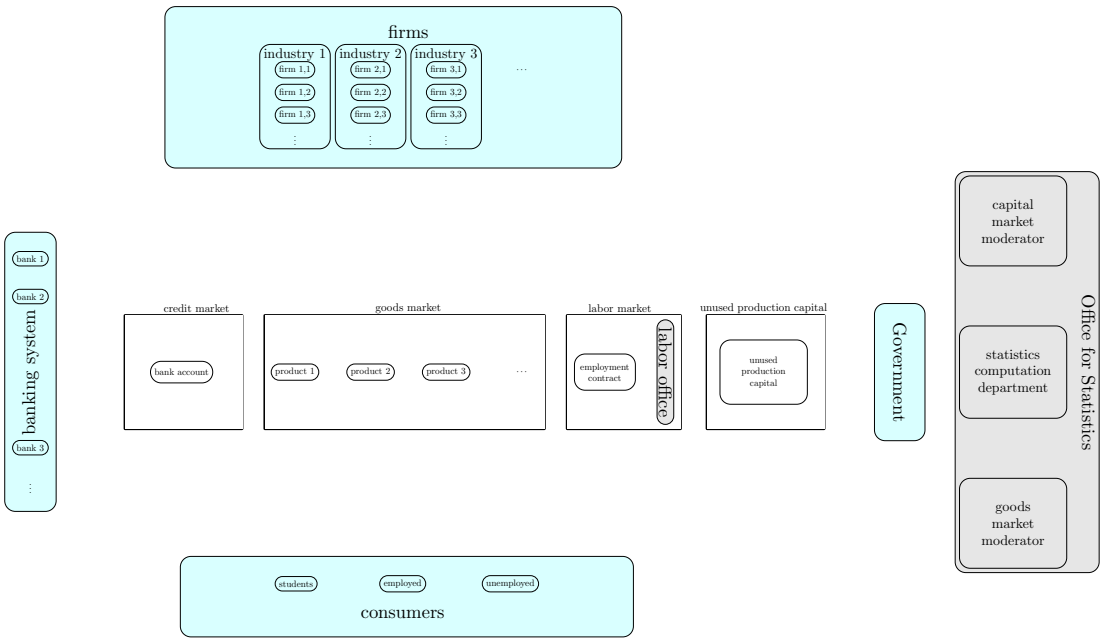


Figure 3.2: Components of the economic system: institutional sectors and other single-instance agents.

3.2 Goods

In this section we focus on the real part of the model while the financial part is treated in the next section.

In the present version of the model, the real part of the economy consists in exchanges of the following items:

- consumption goods
- production inputs
  - investment goods
  - labor

These items are exchanged in the three corresponding markets highlighted in gray in figure 3.3.

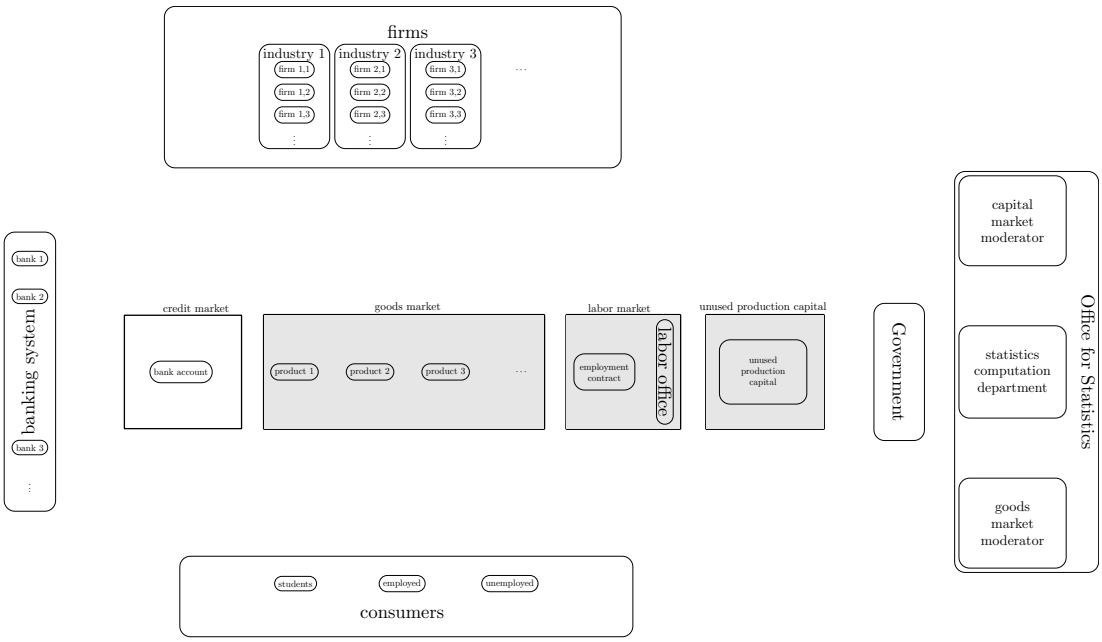


Figure 3.3: Components of the economic system: real markets.

As explained above, the model allows for product differentiation. It concerns consumption products. We have signaled this possibility by indexing the products in the goods market.

Firms production capacity can be increased in two ways: by buying existing unused production capital or by buying newly produced investment goods. To avoid the complication of including a specific industry producing investment goods, the following device is used. Investment goods are realized by assembling

the existing consumption goods. Therefore, demand for new investment goods increases the demand in the consumption goods markets. Once assembled, these goods becomes production capital, and are accounted as assets in the firms balance sheet. Production capital used in the production process gradually depreciates. It can happen that a firm has an excess of production capital (especially in case of a decreasing demand). In this case, the firm can decide to offer the unused production capital on the corresponding market. Firms having shortages of production capital, first check for its availability on the market for unused production capital. If, after these exchanges, additional production capital is needed, the firm asks for new product in the consumption goods market. The latter are then assembled to obtain new production capital. On the other hand, it can happen that the amount of existing unused production capital is larger than the amount asked on the market. In this case, the exceeding production capital that cannot be sold gradually depreciates at a depreciation rate that can be set at a different level to that wearing away the production capital employed in the production.

Finally, to realize the production, labor is needed. The labor market completes the set of real markets of this model.

More details on the markets functioning are given in the next part of the manual.

3.3 Financial assets

In the previous section we have identified the type of goods in our model. The identification and specification of financial products to be included in the model are equally important and are performed in this section.

In the present version of the model, we include the financial contract which is the cornerstone of nowadays financial systems: the bank account.

In this model, every consumer or firm has at leas one bank account. The bank account is a convenient modeling device: it can be either used to store wealth or as a means to obtain credit. In the latter case, the amount on the bank account will be negative.

Having no other financial contracts, in the present version of the model we have intermediated finance only. Financial exchanges are thus performed in the credit market (highlighted in gray in figure 3.4)

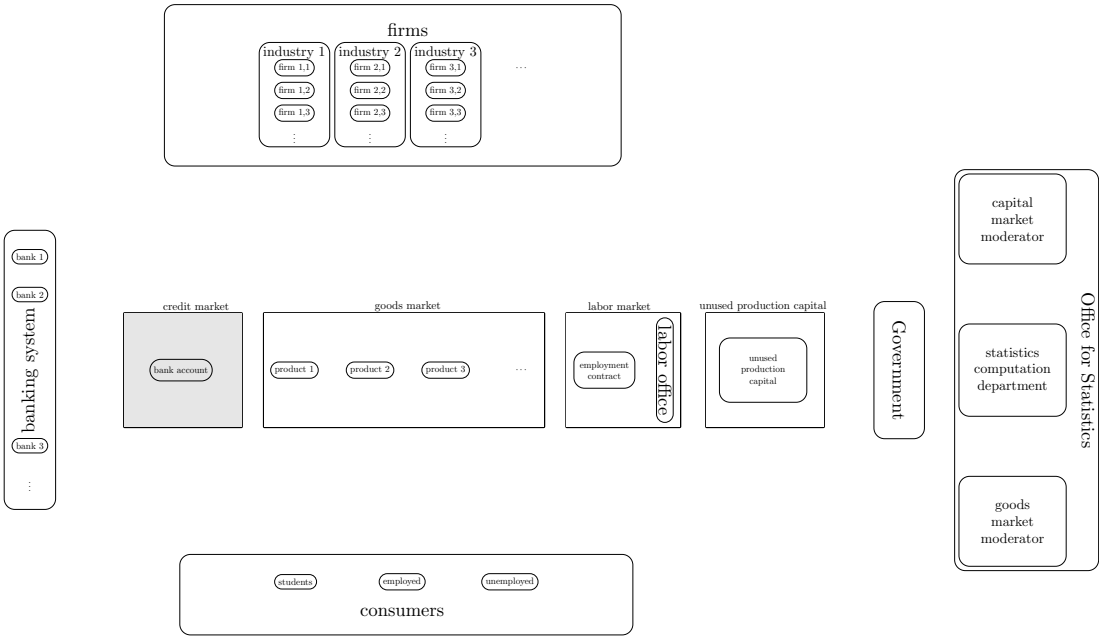


Figure 3.4: Components of the economic system: the credit market.

Considering additional financial contracts, and therefore adding other financial markets to the model, is a goal for the near future. This enrichment will allow the model gradually approach the reality of nowadays financial markets.

Chapter 4

The Dynamics of Events

We start the description by the visual representation of the dynamics of events given in figure 4.1. The figure shows that after the initialization phase (that will be detailed in section 4.1) the software enters in the main loop (deeply discussed in section 4.2). Figure 4.1 gives a brief description of the various events together with the information on the agent(s) involved in them. To save space we use the following abbreviations:

- OFS: office for statistics
- G: Government
- CB: Central Bank
- F: firms
- C: consumers
- C_S: the subset of consumers that are students
- C_E: the subset of consumer that are employed
- C_U: the subset of consumer that are unemployed
- B: banks
- LM: labor market

When an event implies a flux from an agent to another, an arrow is added to give this information. As an example,
$C \rightarrow B$: pay back if possible
stands for: Consumers refund banks if possible.

The initialization and the main loop are detailed in the following subsections.

Figure 4.1: sequence of events

4.1 Initialization

The initialization phase aims at setting firms' potential production. After this step, the sequence of events can progress by entering the main loop. During the initialization process, the stock variables of all agents are set in a consistent way.

Stock Consistency

The details on the initialization process are given hereafter.

To realize production firms need two production input: capital and labor. We use the following Leontief type production function:

$$Y_f = \min(Y_f^{PK}, Y_f^{PL}) \tag{4.1}$$

According to this function, production realized by firm f, Y_f, is the minimum between the potential production that can be realized with the outstanding production capital without labor constraints, Y_f^{PK}, and the potential production that can be obtained by the labor force employed by the firm without capital constraint, Y_f^{PL}.

The capital potential production is assumed to be equal to the level of capital (K_f):

$$Y_f^{PK} = K_f$$

Computing the potential production of labor is more tricky because workers are heterogeneous and have different productivities. We will explain in details how workers' productivities are set in the next section when the consumers initialization will be discussed. For now, it is enough to know that labor is heterogeneous and that a worker's productivity will be identified with $\psi_{w,f}$. To compute the potential production of the workforce employed in a firm, the sum of workers productivities is first computed:

$$\psi_f = \sum_{w \text{ working in } f} \psi_{w,f}$$

Then, we introduce the following parameter:

notation in equations	name in code	value	read from file
θ_{YL}	`parameterOfProductivityInProductionFuncion`	100	yes

The potential production of labor is finally computed as:

$$Y_f^{PL} = \theta_{YL}\psi_f.$$

To arrive at the very first Y_f, several preliminary steps, such as initializing the working state of each consumer, are needed. Some of these steps call for additional operations such as setting up the various agents balance sheets and checking for their consistency. In the remainder of this section we sketch the process, while more details will be given in the following sections.

We adopt the following strategy to set Y_f^{PL} and Y_f^{PK}:

- compute the potential output from workers
- adjust capital such as the potential output from capital is equal to the potential output from workers

As already mentioned, the first step implies a preliminary setup of household working state. The second step, instead, allows the setup of firms assets balance sheet item.

Knowing firms balance sheet assets, we set the liabilities in such a way that the balance sheet identity is satisfied.

Next, we initialize the household balance sheet.

Once the amount of households' and firms banks accounts are known, we setup banks balance sheets.

So the whole initialization sequence has the following phases:

1. create agents;
2. set consumers working state;
3. compute each firm labor potential production and set the production capital to a level that allows a potential production equal to that of labor;
4. set agents balance sheet;
5. revise agents balance sheet for stock consistency.

Stock Consistency

These phases are explained in details in the technical documentation. There, we present the steps taken for initializing the model in a sequential order. We also document the above mentioned phases by using UML activity diagrams.

Readers that would not delve into the technical documentation at this point can progress to the following sections where we discuss deeply the initialization for each type of agent.

4.1.1 Consumers initialization

Working situation

Students

Each worker is characterized by its fundamental ability. It characterizes each consumer performance during the education period. The latter in turn determine the productivity as a worker.

We call this parameter the ability (as) student and denote it $\theta_{c,as}$. We set it during initialization by drawing from a uniform distribution Θ_{as}:

$$\Theta_{as} \sim U(\theta_{minas}, \theta_{maxas})$$

The following parameters are thus used:

notation in equations	name in code	value	read from file
θ_{minas}	`Context.minAbilityStudent`	0.35	yes
θ_{maxas}	`Context.maxAbilityStudent`	0.5	yes

The most important observation about this parameter is that its upper value is 0.5.

Another important variable is the consumers age, which is initialized as a random number between zero and the parameter θ_{cea} denoting the retirement age:

notation in equations	name in code	value	read from file
θ_{cea}	`Context.consumerExitAge`	70	yes

Using the `abilityStudent` and the `consumerAge` the education history of consumers is initialized. Each year of education is set to successful if that year uniform random draw u is less that two times the student ability:[1]

$$u < 2\theta_{c,as}$$

and unsuccessful otherwise. Note that best students has $\theta_{c,as} = 0.5$, so they will be alway successful because $2\theta_{c,as} = 1$. Students with less abilities has lower probability to be successful.

To initialize the education history we use the following parameters:

notation in equations	name in code	value	read from file
θ_{mnfpe}	`maxNumberOfFailedPeriodsOfEducation`	2	yes
θ_{mnpe}	`maxNumberPeriodsOfEducation`	21	no

Starting from age 0, the process is repeated until

1. the maximum number of failures admitted (θ_{mnfpe}) or
2. the maximum periods of possible education ($\theta_{mnpe} + \theta_{mnfpe}$) or
3. the consumer's age

[1] Note how $2\theta_{c,as} \leq 1$ and u is drawn from a $U(0, 1)$.

is reached.

We then count the number of successful periods of education, $n_{c,sye}$.

When the process is stopped by conditions 1 or 2, the consumer state is set to non students. At this stage, all non students are unemployed. This state will be revised in the next step. Both the education degree and the productivity as a worker are assigned using $n_{c,sye}$.

The degree of education is assigned as follows:

$n_{c,sye}$	degree	degree id
$0 \leq n_{c,sye} < 5$	none	0
$5 \leq n_{c,sye} < 8$	elementary	1
$8 \leq n_{c,sye} < 13$	intermediate	2
$13 \leq n_{c,sye} < 16$	college	3
$16 \leq n_{c,sye} < 18$	bachelor	4
$18 \leq n_{c,sye} < 21$	master	5
$21 = n_{c,sye}$	PhD degree	6

The productivity is assigned as follows. Each year of education increases the consumers ability by a fixed amount. So, the productivity (ψ_c) assigned to a consumer with $n_{c,sye}$ successful year of education is

$$\psi_c = \theta_{c,as} + 0.5\frac{n_{c,sye}}{\theta_{mnpe}}$$

Note that $\theta_{c,as} = 0.5$ implies $n_{c,sye} = \theta_{mnpe}$ so that $\psi_c = 1$.

Once the productivity of a non student is known, we compute his/her potential production Y_c^P as the product of θ_{YL} and ψ_c:

$$Y_c^P = \theta_{YL}\psi_c$$

The education history initialization is stopped by condition 3 when the consumer is young. In this case the subject is assigned the state of student and its education history will be evolved in the main loop using the rules explained above.

Employed and unemployed

At this stage all non students are unemployed. Now, some of them will be employed by firms.

To perform this task we introduce the following parameter:

notation in equations	name in code	value	read from file
θ_{ptbub}	`Context.probabilityToBe UnemployedAtTheBeginning`	0.2	yes

With probability $1 - \theta_{ptbub}$, each non student selects a firm randomly and send his/her CV to this firm. Subjects who do not send a CV (this happens with probability θ_{ptbub}) enter the main loop in the unemployment state.

Initializing consumers bank account

This is the first action performed to setup the consumers balance sheet. Other actions will contribute to the formation of the consumers' bank account.

The involved parameters are:

notation in equations	name in code	value	read from file
θ_{nbcc}	`Context.numberOfBanksAConsumerCanBeCustumerOf`	1	yes
θ_{mincba}	`Context.minConsumerInitialBankAccount`	-500	yes
θ_{maxcba}	`Context.maxConsumerInitialBankAccount`	500	yes

Each consumer selects randomly a number of banks equal to θ_{nbcc} and open a bank account in each of them. To set the amount of each bank account the code draw a random integer from a uniform distribution $U(\theta_{mincba}, \theta_{maxcba})$. In case the figure is negative the amount is set to zero and the drawn number is assigned to the `demandedCredit` variable (it will be managed later in the initialization process). Non negative number are assigner to the bank account amount.

As hinted at above, the amount of the various bank accounts will be revised in the final step of the initialization to ensure all the agents' balance sheet consistency.

Stock Consistency

4.1.2 Firms initialization

Potential production of labor

In the consumers initialization we saw that some non student sent their CV to firms. So, each firm now has a list of CVs. Each firm employs all the senders of the CVs that are in the received CVs list. The CV senders state is switched from unemployed to employed.

As already explained at the beginning of this section, the firm determines the potential production of its labor force (ψ_f) by summing the potential production of each employee ($\psi_{w,f}$):

$$\psi_f = \sum_{w \text{ working in } f} \psi_{w,f}$$

the potential production is thus

$$Y_f^{PL} = \theta_{YL}\psi_f$$

Production capital

As we told above, due to the Leontif production function, the production capital is set in such a way that the potential production from capital is equal to the

potential production of labor. We have already defined the potential production of capital as

$$Y_f^{PK} = K_f$$

So, the initial level of capital (K_f) for each firm is set to

$$K_f = Y_f^{PL}$$

The level of production

Now that the levels of both production inputs are known, the production made by each firm is computed by using equation (4.1).

Initialization of firms balance sheet

In the previous step we set up the assets side of the firms balance sheet that is equal to the production capital.

As explained above, we have two liabilities: equity and bank account. We set up equity and determine the bank account residually.

To setup equity we use the following parameters:

notation in equations	name in code	value	read from file
$\theta_{minfier}$	`Context.minFirmInitialEquityRatio`	0.1	yes
$\theta_{minfier}$	`Context.maxFirmInitialEquityRatio`	0.3	yes

Each firm equity base (E_f) is set as follow

$$E_f = u_{Ef} K_f$$

where u_{Ef} is drawn from $U(\theta_{minfier}, \theta_{maxfier})$.

Finally, each firm sets her debt (B_f) by using the balance sheet identity:

$$B_f = K_f - E_f$$

As specified above, debt is obtained through bank accounts. In this version of the model a firm is costumer of a number of banks given by the following parameter:

notation in equations	name in code	value	read from file
θ_{nbfc}	`Context.numberOfBanksAFirmCanBeCustumerOf`	1	yes

Each firm selects randomly a number of banks equal to θ_{nbfc} and open a bank account in each of them. The amount of the bank account (BA_{fb}) is set to

$$BA_{fb} = -\frac{B_f}{\theta_{nbfc}}$$

4.1.3 Banks initialization

We remember that in this version of the model, we have only one financial contract: the Bank account (BA). Bank accounts can have a positive amount or a negative one. Hereafter, we denote a bank account with a positive amount with BA^+ and BA^- denotes a negative amount.

To setup the banks balance sheet, we start again from their assets. Bank assets are households' and firms financial liabilities. In this version of the model, financial liabilities are given by negative bank accounts. The sum of negative bank accounts gives the loans extended by a bank (L_b):

$$L_b = \sum_f BA^-_{fb} + \sum_c BA^-_{cb}$$

Given assets of a bank, we setup the its equity (E_b) by using the following parameters:

notation in equations	name in code	value	read from file
$\theta_{minbier}$	`Context.minBankInitialEquityRatio`	0.1	yes
$\theta_{maxbier}$	`Context.maxBankInitialEquityRatio`	0.3	yes

A bank equity base is then computed as

$$E_b = u_{Eb} L_b$$

where u_{Eb} is drawn from $U(\theta_{minbieb}, \theta_{maxbieb})$.

Now, the level of deposits (D_b) compatible with the balance sheet relationship is computed:

$$Dep_b = L_b - E_b$$

Due to the random elements in the initialization of consumers and firms, the following stock inconsistency is expected at this stage of the initialization:

$$\sum_{c \text{ customer of } b} BA^+_{cb} \neq Dep_b$$

Note that only consumers' positive bank accounts are considered because firms have no deposits at initialization.

The following revision is performed to restore stock consistency. The amount of each positive bank account is adjusted as follows:

$$BA^+_{cb} = BA^+_{cb} \frac{Dep_b}{\sum BA^+_{cb}}$$

In this way, stock consistency in each bank balance sheet is restored:

$$\sum BA^+_{cb} = Dep_b$$

Stock Consistency

4.1.4 Government and the Central Bank

Because we have only the bank account as financial contract, the Government can finance its deficit by borrowing on its bank account at Central Bank.

The initialization consists thus in opening the Government bank account at Central Bank and setting its amount. This is the initial level of the public Debt.

Because the aggregate production has not been yet computed, we link this amount to an estimation of this variable.

$$Y_0^e = C(1 - \theta_{ptbub})0.5\theta_{YL}$$

where $C(1 - \theta_{ptbub})$ is an estimation of the number of workers and $0.5\theta_{YL}$ an estimation of the average productivity of workers.

4.1.5 Conclusions

At the end of the initialization phase the following results are achieved.

Consumers know if they are student, employed or unemployed. Non students know their level of productivity. Employed know in which firm they are working.

Firms knows their employees and the level of their production capital. The supply of goods is thus computed.

All the agents are also endowed with a balance sheet. We think it useful to look at the following figure which gives a visual representation of the balance sheet at the sector level. The notation is similar to that used above, but capital letters in lower scripts are used to denote sectors. So, F denotes the production sector, B the banking sector and C the consumers sector.

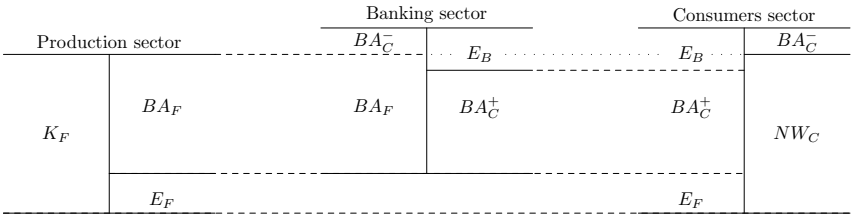


Figure 4.2: balance sheets of the various sectors and their relationship

The figure also highlights the coherence of the balance sheets and what this coherence implies. As one can easily check, the net wealth of the consumers sector (NW_C) is equal to production capital (K_F). Second, the figure shows that equities of banks and firms make up consumers wealth together with the more straightforward item represented by deposits.

We pointed out above that the bank account is the sole financial contract in this model. The accounting is thus performed at the individual level for the wealth stored in bank accounts. Equities are not represented by shares, thus we have not a measure of individual wealth hold in shares. The amount of wealth that is represented by equity is computed at the aggregate level. These aggregate variables could be taken into account by consumers when taking decisions.

4.1.6 Technical documentation

People who need more details can fint them in the technical documentation.

4.2 The main loop

The main loop can be divided into two parts:

1. the process that leads to the production of goods
2. the process that leads to the consumption of what has been produced

The parts of the main cycle where these two processes develop are highlighted with colors: in red the consumption process and in violet the production one. The figure also highlights which markets are interested by the two processes.

4.2.1 The process that leads to the consumption of what has been produced

We present here the sequence of events and will enter in the details of each item in the following subsections.

The reader can easily check the correspondence between the numbering of events in figure 4.3 and that in the following list. The sequence of events is the following:

1. workers receive wage;
2. Office for statistics, Government and Central Bank implement fiscal policy
   - Office for statistics provide data
   - Central Bank sets government borrowing limit
   - Government decides the tax rate
   - workers pay taxes and unemployed receive the dole
   - The government bank account at Central Bank is updated
3. banks
   - account interest
   - ask for loan repayments to indebted consumers;
4. consumers refund if possible (if they have enough financial resources);
5. students update their state
6. consumers compute desired consumption and ask for credit in order to achieve the desired level
7. banks decide how much credit to allow;
8. Allowed consumption is computed: consumers adjust their desired consumption according to allowed credit.
9. Goods markets open: Office for statistics allocate demand

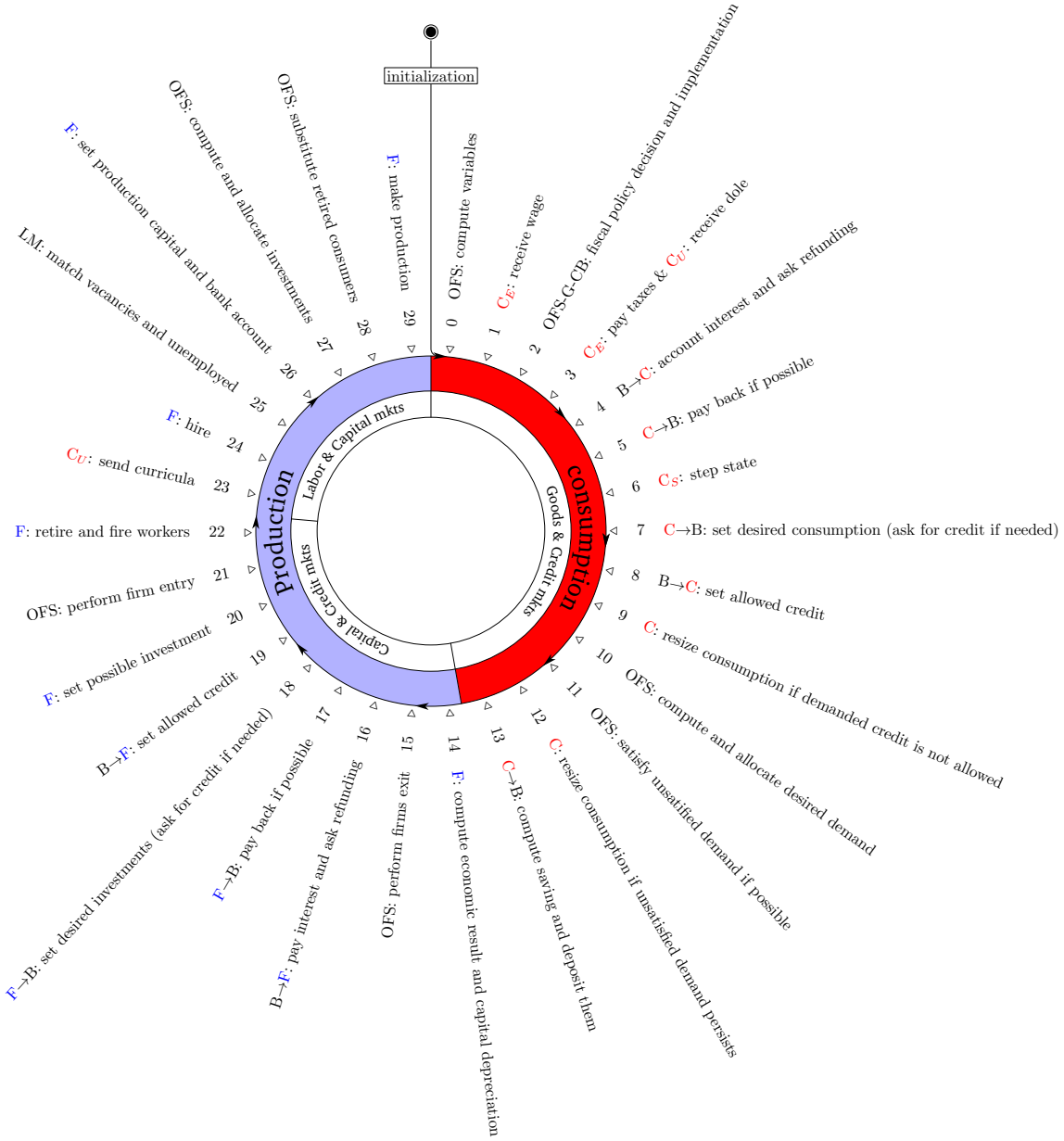


Figure 4.3: sequence of events

10. Office for statistic satisfies unsatisfied demand if possible.
11. The *effective* level of consumption is established: it may be less or equal to the allowed consumption.
12. consumers adjust their bank accounts (deposits and loans) according to effective consumption.

As for the initialization, these phases are explained in details in the technical documentation. There, we present the steps of the main loop in a sequential order and each step is detailed by means of an UML activity diagrams.

We will present hereafter the steps in less schematic more verbal way. In any case, the UML diagrams can be visualized by a link provided at the end of each paragraph.

Consumers receive wages

The computation of workers' wage is less straightforward. The wage each employed consumer receives is a mark up on the unemployment dole:

$$w_w = ud + \theta_{LM}\theta_{YL}f(\psi) \tag{4.2}$$

where:

notation in equations	name in code	value	read from file
ud	`Context.unemploymentDole`	30	yes
θ_{LM}	`Context.laborMarketStateToSetWage`	0.5	yes

θ_{YL} is the parameter of productivity in production function (already defined) and $f(\psi)$ is a function of workers' productivity.

Three possibilities for $f(\psi)$ are presently implemented:

1. $f(\psi)$ is the worker's productivity;
2. $f(\psi)$ is average productivity of workers having the worker's degree in the employer's firm;
3. $f(\psi)$ is average productivity of workers having the worker degree in the economy.

This leads to three different wage setting rules. One of these rules can be chosen by means of the following parameter:

notation in equations	name in code	value	read from file
wsr	`Context.wageSettingRule`	2	yes

The researcher can choose which rule to use in a given run by setting the `value` of this parameter in the `batch_parameters.xml` file located in the scenario folder. The possible values that can be specified in the parameter file are: 0, 1 and 2. Their meaning is as follows:

- If `value=0` the wage depends on the worker's productivity.
- If `value=1` the wage depends on the productivity of workers employed in the firm having the same education level of the considered worker.
- If `value=2` the wage depends on the productivity of workers having the same education level of the considered worker in the whole economy.

The UML sequence diagram of this event is available here.

Office for statistics, Government and Central Bank implement fiscal policy

In this step the Office for statistics, the Government and the Central Bank interact in a process that will lead each consumer to know his/her disposable income.

First of all, the Office for statistic computes the figures needed to compute the public balance: the aggregate taxable income and the aggregate dole to be payed.

Second, the central bank sets the amount of funds that is willing to borrow to the government.

Third, the government sets the tax rate that will be charged to workers, wile the unemployment dole is kept at the consumption subsistence level.

Finally, workers pay taxes and unemployed receive the dole.

Office for statistics compute the taxable income and the number of unemployed

The aggregate taxable income is given by the sum of wages:

$$TI = \sum_{w} w_w$$

The number of unemployed is denoted with U.

The central bank

First of all the Central Bank charge the interest rate on public debt:

$$PD = PD(1 + i^{-})$$

Then, the Central Bank decides on the credit allowed to Government using the public debt-production ratio:

$$pd = \frac{PD}{Y}$$

The outcome of the process can be:

- $pd\uparrow$: the Government can increase the debt but the new pd cannot be higher than $pd + dpd$;
- $pd =$: the Government cannot increase its debt;
- $pd\downarrow$: the Government must reduce the pd at least to $pd + dpd$.

These events happen with the following probabilities:

$$\begin{array}{lcl} pr(pd\uparrow) & = & 0.5e^{-0.69pd} \\ pr(pd =) & = & pr(pd\uparrow) \\ pr(pd\downarrow) & = & 1 - pr(pd\uparrow) - pr(pd =) \end{array}$$

The following table provides some numerical examples

pd	$pr(pd\uparrow)$	$pr(pd =)$	$pr(pd\downarrow)$
0	0.5	0.5	0
0.3	0.4	0.4	0.2
0.6	0.33	0.33	0.33
2.0	0.126	0.126	0.748

It can be easily checked that the probability that the central bank finances a public deficit decreases with the level of pd while the probability that the Central Bank insists on the realization of a public surplus increases with pd.

The outcome of the process is thus the pd desired by the central bank:

$$pd^d = \begin{cases} pd + dpd & \text{with probability} \quad pr(pd\uparrow) \\ pd & \text{with probability} \quad pr(pd =) \\ pd - dpd & \text{with probability} \quad pr(pd\downarrow) \end{cases}$$

Thus, we can compute the desire public debt as

$$PD^d = pd^d Y$$

and the desired public surplus as

$$PD - PD^d$$

The government

The government decides the tax rate taking account of the Central Bank recommendations. However, as we will explain below, sometimes the government can oppose Central bank.

Fist of all, the government computes the tax rate that satisfies the Central Bank recommendation by solving τ^d in the following equation:

$$PD - PD^d = \tau^d \; TI - d \; U$$

This gives

$$\tau^d = \frac{PD - PD^d + d \; U}{TI}$$

Setting the tax rate is a little bit tricky. First of all, we observe that the government can always set a tax rate higher than that desired by the central bank. Because we exclude that the government increases the consumers' sacrifice, this possibility is applied only when $\tau^d < 0$. In this case the government set $\tau = 0$ and uses only a part of the financial resources the Central Bank is willing to lend.

In the other case, τ is set to τ^d except when τ^d is judged too high and the Government can enforce a lower tax rate. The Central bank suggestion can be opposed by rejecting its request to be refunded. Fulfilling a Central bank request of refunding, i.e. reducing the public debt, often causes an increase in the tax rate. When this increase is judged too high, the government can contain the increase by partially or totally denying The Central Bank request. In this case, an unpaid amount will be recorded for the government.

To model this possibility, we introduce a tax rate that the government would not exceed, $\bar{\tau}$. We then compute the tax rate that would prevail with no amount refunded:

$$\tilde{\tau} = \frac{d\ U}{TI}$$

Note that when refunding is asked, $\tau^d > \tilde{\tau}$. Now, when $\tilde{\tau} > \bar{\tau}$ the government cannot reduce the tax rate at the desired level $\bar{\tau}$ and it is forced to sets the tax rate to $\tilde{\tau}$. When $\tilde{\tau} < \bar{\tau} < \tau^d$ the government partially fulfills the Central Bank requests by setting the tax rate to $\bar{\tau}$.

So, the tax rate is set as follows:

$$\tau = \begin{cases} 0 & \text{if} \quad \tau^d < 0 \\ \tau^d & \text{if} \quad \tilde{\tau} > \tau^d \\ \tilde{\tau} & \text{if} \quad \bar{\tau} < \tilde{\tau} < \tau^d \\ \bar{\tau} & \text{if} \quad \tilde{\tau} < \bar{\tau} < \tau^d \end{cases}$$

Consumers adjust to fiscal policy

Now, the government collects taxes and pays unemployment dole (ud). The consumers can therefore compute their available income.

Workers' vailable income is given by $w_w(1 - \tau)$.

Unemployed, receive the dole and pay no taxes, therefore their available income is equal to the dole.

Students have no income so their available income is equal to zero.

Banks account interests and ask for loan repayments

The accounting of interest updates the households bank accounts. This is done at different interest rates according to the sign of the bank account (BA_{cb}).

Positive bank accounts are updated using the interest rate on deposits (i^+), so that if $BA_{cb,t} \geq 0$

$$BA_{cb} = BA_{cb}(1 + i^+)$$

The model has two different interest rates on loans. The ordinary interest rate (i^-) which is charged on bank accounts with negative amount owned by workers, and the subsidized interest rate i^-_{sub}, charged on negative bank accounts owned by unemployed people and students. Thus, a negative bank account hold by a workers is updated as follows

$$BA_{cb} = BA_{cb}(1 + i^-)$$

while if the bank account belongs to a student or an unemployed we will have

$$BA_{cb} = BA_{cb}(1 + i^-_{sub})$$

Once the accrued interest has been accounted, the bank may ask the reduction of some negative accounts. This involves only the accounts owned by workers.

The amount of a negative bank account owned by a worker desired by the bank (BA^{db}) is determined as follows:

$$BA^{db} = \begin{cases} BA(1-\theta_{is}) & \text{with probability } pr_{is} \\ BA & \text{with probability } 1 - pr_{is} \end{cases}$$

The parameters involved in this computation are:

notation in equations	name in code	value	read from file
i^+	`Context.interestRateOnDeposits`	0.001	yes
i^-	`Context.interestRateOnLoans`	0.004	yes
i^-_{sub}	`Context.interestRateOnSubsidizedLoans`	0.001	yes
θ_{is}	`Context.percentageOfLoanToRefundFor IndebtedWorkersIfAsked`	0.1	yes
pr_{is}	`Context.probabilityToBeAskedToRefund ForIndebtedWorkers`	0.5	yes

The UML sequence diagram of this event is available here

Consumers refund if possible

In the previous step banks can ask to reduce amount of negative bank accounts.

In this model households can have more than one bank account. In principle, some of them can be positive, and others negative. The downward adjustment can be asked on some of the negative accounts. Households can face banks refunding request using all their financial assets: income (wage) and their positive bank account.

Households first try to fulfill bank requests by moving funds from positive to negative bank accounts. If this is not enough, they use their income. However, income can be used to fulfill banks requests as long as its amount is not lower than the subsistence level of consumption. In the worst case where banks request

cannot be satisfied by using all the available financial resources, the household consumes the subsistence level of consumption and the bank reduces the negative amount of the bank account by an amount lower than it would and consequently record a positive unpaid amount.

The UML sequence diagram of this event is available here

Students update their states

The evolution of a student state follows the process already described in the initialization (see page 36).

In particular, the current period of education is successful if the new uniform random draw u is less that two times the student ability:

$$u < 2\theta_{c,as}$$

and unsuccessful otherwise.

If the education period is successful, the number of successful years of education ($n_{c,sye}$) is increased:

$$n_{c,sye} = n_{c,sye} + 1$$

and the degree of education is updated according to the table already reported above:

$n_{c,sye}$	degree	degree id
$0 \leq n_{c,sye} < 5$	none	0
$5 \leq n_{c,sye} < 8$	elementary	1
$8 \leq n_{c,sye} < 13$	intermediate	2
$13 \leq n_{c,sye} < 16$	college	3
$16 \leq n_{c,sye} < 18$	bachelor	4
$18 \leq n_{c,sye} < 21$	master	5
$21 = n_{c,sye}$	PhD degree	6

The subject's productivity is the undated as explained in the initialization section:

$$\psi_c = \theta_{c,as} + \frac{0.5}{\theta_{mnpe}} n_{c,sye}$$

The subject looses his/her state of student

1. in case of a success, if the maximum periods of possible education (θ_{mnpe} + θ_{mnfpe}) or

2. in case of failure, if the maximum number of failures admitted (θ_{mnfpe})

is reached. In these cases, his/her status changes in unemployed.

The UML sequence diagram of this event is available here

Consumers compute desired consumption and ask for credit in order to achieve the desired level

This task is performed by taking the following steps.

Each consumer check her/his income: workers have the wage as an income, unemployed have the dole and students have no income. Consumers also computes their financial resources by looking at their bank accounts.

Based on this information the desired consumption is computed using a consumption function. In the current version of the model this amount is computed randomly and it can be higher or lower than the available income.

This allows to compute the desired total level of expenditure of consumption: D_c^d.

In this model we can have product differentiation. When more than one product are present, the desired demand for each item ($D_{c,j}^d$) is computed as follows:

$$D_{c,j}^d = \alpha_{c,j} D_c^d$$

where $\alpha_{c,j}$ is an index of the consumers appreciation for the given item. Of course the condition $\sum_j \alpha_{c,j} = 1$ must hold.

If the available financial resources are not enough to realize desired consumption, consumers evaluate the possibility to ask for credit. This possibility is precluded if all the consumers' bank accounts have a positive unpaid amount. Otherwise, the consumer asks all the credit needed to the bank with the best bank account.

Credit is also asked to the bank with the best bank account to satisfy unpaid amounts that exist in other banks.

The UML sequence diagram of this event is available here.

Banks decide how much credit to allow

In the previous step, consumers who need credit set its desired amount on one of their bank accounts (the best bank account) having null unpaid amount if such account exists. Let us identify this variable with BA_c^d (where the d upper script means **d**esired). Because they are asking for credit, this amount is negative: $BA_c^d < 0$. Note that the best bank account has either a positive or a negative amount.

In the present version of the model, the bank decides the allowed credit $BA_c^a < 0$ (superscript a means **a**llowed) as follows

- if $BA_{cb} \geq 0$ and $BA_c^d < 0$

$$BA_c^a = \begin{cases} BA_c^d & \text{with probability} \quad pr_{ab} \\ \theta_{ab} BA_c^d & \text{with probability} \quad 1 - pr_{ab} \end{cases}$$

- if $BA_{cb} < 0$ and $BA_c^d < BA_{cb}$

$$BA_c^a = \begin{cases} BA_c^d & \text{with probability} \quad pr_{ab} \\ BA_{cb} - \theta_{ab}(BA_{cb} - BA_c^d) & \text{with probability} \quad 1 - pr_{ab} \end{cases}$$

This does not apply to students to whom the asked credit is always allowed.

We have thus introduced the following parameters:

notation in equations	name in code	value	read from file
θ_{ab}	`Context.percentageOfCreditAllowedTo ConsumersWhenCreditIsNotTotallyFunded`	0.0	yes
pr_{ab}	`Context.consumersProbabilityToGetFunded`	0.5	yes

The UML sequence diagram of this event is available here.

Consumers adjust their desired consumption according to allowed credit

Because there is the possibility that bank does not allow all the demanded credit, we introduce a new variable: the allowed demand (D_c^a).

This variable can differ from the desired demand only for consumers who asked for credit an whose request was not satisfied by banks.

Because we can have product differentiation, we first compute the allowed-desired demand ratio on the whole consumption expenditure:

$$r_c^d = \frac{D_c^a}{D_c^d}$$

Then we scale the demand of each single item by this amount

$$D_{c,j}^a = r_c^d D_{c,j}^d$$

In this resizing, we do not account of unpaid amounts because we assume that credit is first used to satisfy consumption. The possible residual is then used to reduce unpaid amounts.

The UML sequence diagram of this event is available here.

Good markets open: office for statistics computes and allocate desired demand and satisfy unsatisfied demand if possible

We code the goods market functioning in such a way that both centralized and decentralized matching mechanisms can be mimicked. In particular, the Office for Statistic supervise the allocation of demand.

The task is performed by taking the following steps:

- compute the desired allowed demand for each good type;
- allocate it to firms;
- manage unsatisfied demand;
- update firms sales.

These items are detailed hereafter.

• *Compute the allowed demand for each good type.* The Office for Statistics computes the demand of each industry by summing over all the consumers. We mimic decentralized markets and the losses of exchange opportunities due to the matching process by introducing the following parameter:

notation in equations	name in code	value	read from file
θ_{gmi}	`Context.percentageOfDemandMissed BecauseOfGoodsMarketImperfections`	0.0	yes

The demand of industry producing item j is therefore computed as follows:

$$D^a_{j\leftarrow c} = (1-\theta_{gmi}) \sum_c D^a_{c,j}$$

where the $\leftarrow c$ signal that demand comes from consumers. This notation allows to identify the various sources of demand that will be introduced below in the text and in future extensions.

Provided that in the economy there are J industries, the software computes an array of dimension J: $\{D^a_{1\leftarrow c}, \cdots, D^a_{J\leftarrow c}\}$.

The desired allowed aggregate demand is also computed:

$$D^a_{\leftarrow c} = \sum_j D^a_{j\leftarrow c}$$

The UML sequence diagram of this event is available here.

• *Allocate desired allowed demand to firms.* In this step, the demand computed in the previous step ($D^a_{j\leftarrow c}$) is allocated to the various firms according to their share of production in the industry they belong to:

$$D^a_{f,j\leftarrow c} = D^a_{j\leftarrow c} \frac{Y_{f,j\rightarrow c}}{Y_{j\rightarrow c}} \tag{4.3}$$

It will be clarified in the `make production` step that firms produce goods that are sold to other firms ($Y_{f,j\rightarrow f}$) to let them adjust their production capital and consumption good sold to consumers ($Y_{f,j\rightarrow c}$). The just written computation involves the latter.

The specific step we are discussing allows firms to know the appreciation of the item they produce among consumers. Note that firms in some industries may be in a short supply situation ($D^a_{j\leftarrow c} > Y_{f,j\rightarrow c}$) while in other industries the opposite may hold. This information will be used by firms to guide their production choices.

The UML sequence diagram of this event is available here.

• *Manage unsatisfied demand.* In this model we have vertical differentiation. This means that products with higher j are more advanced and, therefore, preferred. It may thus happen that demand on these product is higher than

supply. In case the demand of a given product cannot be entirely satisfied, the office for statistics try to fulfill consumers excess demand of this product with the product that precede it in the quality ladder. The Office for statistic moves demand across industries starting from the most advanced product and proceeding backward. This mechanism aims to mimic consumer decision to buy a less advanced product if the item s/he desires is in short supply.

At the end of this process the demand received by each industry ($D_{j\leftarrow c}$) and that received by each firm ($D_{f,j\leftarrow c}$) is known.

Consumers are then informed of the demand reallocation and their final demand is computed:

$$D_{c,j} = D^a_{c,j} \frac{D_{j\leftarrow c}}{D^a_{j\leftarrow c}}$$

The UML sequence diagram of this event is available here.

- *Update firms sales.* Firms are also informed on the items they have sold:

$$D_{f,j\leftarrow c} = D_{j\leftarrow c} \frac{Y_{f,j\rightarrow c}}{Y_{j\rightarrow c}}$$

This information will be then used to compute profit.

Two UML sequence diagrams describe this event. The first one describe the computation performed by the office for statistics of the demand starting from the consumer data. This allows a double check on the computation of $D_{j\leftarrow c}$ an is available here. The second one describes the allocation of demand to firms and is available here.

Consumers adjust their bank accounts (deposits and loans) according to effective consumption

Summing up, in this model the desired consumption can be resized two times. The first one when the consumer asks for credit to achieve its desired consumption, but the bank does not allow the demanded credit. In this case we have that the allowed consumption (D^a_c) is lower than desired consumption (D^d_c). The second one is when the asked good is in short supply. In this case, we say that the effective consumption (D_c) is lower than the allowed consumption (D^a_c). The effective consumption is computed as follows:

$$D_c = \sum_j D_{c,j}$$

Once the effective consumption is known, consumers pay the due amounts to firms. The consumers financial position is adjusted accordingly. Those who consumes less that their income brings their saving to the bank with the worst position. Those who were allowed credit decrease their best bank account by the amount needed to consume. If additional allowed credit remains, it is used to reduce or cancel the unpaid amounts possibly present in the other bank accounts.

The UML sequence diagram of this event is available here.

4.2.2 Example of consumer-banks relationship

A numerical example of the sequence of action of the above explained process is given hereafter.

The focus is on an indebted Consumer.

We trace hereafter the steps explained in the manual.

Step 1: interests and loans repayment

In this example, each consumer is customer of three banks.

First of all, banks account the interest rate.

Suppose that after accounting interest rate, the consumer's bank accounts have the following amounts:

```
bank 1 account =   10
bank 2 account = -150
bank 3 account =  -50
```

The bank assumes that indebted consumers (those with a negative bank account) ask for the whole renewal of the debt:

```
bank 1 account =   10  demanded credit =    0
bank 2 account = -150  demanded credit = -150
bank 3 account =  -50  demanded credit =  -50
```

Each bank with a negative account can ask for refunding. If this happens, the allowed credit is lower (in absolute value) than the demanded credit. Suppose banks 2 and 3 intend to reduce their exposition as follows:

```
bank 1 account =   10  demanded credit =    0 allowed credit =    0
bank 2 account = -150  demanded credit = -150 allowed credit = -130
bank 3 account =  -50  demanded credit =  -50 allowed credit =  -45
```

In this example, the consumer needs 25 to satisfy banks requests.

step 2: refunding

The consumer refunds her bank account if she has enough income (to refund) and, in any case, refunds an amount that allows a subsistence consumption level. Let's consider this example:

`disposableIncome=40`

and that the subsistence consumption is 10.

The software first computes the resources available to refund. They are given by the sum of positive amounts in bank accounts plus the disposable income minus the subsistence consumption. In this example we have:

`resourceAvailableToRefund = 10 + 40 - 10 = 40`

These resources are enough to satisfy banks' requests; the consumer refunds totally banks because her financial resources allow both debt repayment and a consumption greater than the subsistence level. Therefore, the new situation is:

```
bank 1 account =   10  demanded credit =    0 allowed credit =    0
bank 2 account = -130  demanded credit = -150 allowed credit = -130
bank 3 account =  -45  demanded credit =  -50 allowed credit =  -45

disposableIncome = 15
```

Consider now a slightly different situation in bank accounts:

```
bank 1 account =   15  demanded credit =    0 allowed credit =    0
bank 2 account = -150  demanded credit = -150 allowed credit = -130
bank 3 account =  -50  demanded credit =  -50 allowed credit =  -45
```

the only difference with the previous situation is that the amount in bank 1 is 15 instead of 10. Suppose furthermore that the consumer's income is 15 instead of 40. Now resources available to refund banks (the sum of positive amounts in bank accounts plus the disposable income minus the subsistence consumption) are given by:

`resourceAvailableToRefund = 15 + 15 - 10 = 20`

They are not enough to satisfy banks' requests (25 is needed). Suppose 15 is used to refund bank 1 and 5 to refund bank 2. Unpaid amounts are recorded and disposable income is set to allow the subsistence consumption:

```
bank 1: account =    0  demanded credit =    0 allowed credit =    0 unpaid =  0
bank 2: account = -135  demanded credit = -150 allowed credit = -130 unpaid =  5
bank 3: account =  -50  demanded credit =  -50 allowed credit =  -45 unpaid =  5

disposableIncome = 10
```

We continue this example with this second situation.

Step 3: account resetting

In this step, banks set the demanded and allowed credit to zero.

Banks accounts are the updated as follows:

```
bank 1: account =    0  demanded credit =  0 allowed credit = 0 unpaid =  0
bank 2: account = -135  demanded credit =  0 allowed credit = 0 unpaid =  5
bank 3: account =  -50  demanded credit =  0 allowed credit = 0 unpaid =  5

disposableIncome = 10
```

step 4: consumers set desired credit

Now each consumer can asks for new credit. This can be done for two reasons: 1) to achieve a desired consumption higher than disposable income and 2) to pay unsatisfied lenders.

Suppose now that the consumer would like to consume 20.

She asks for additional financial resources both to finance consumption (10) and to pay back bank 2 and 3 (5+5).

The additional credit amount of 20 is asked to one bank. In particular, the bank with the "best" account (bank 1) is chosen.

```
bank 1: account =    0  demanded credit = -20 allowed credit = 0 unpaid =  0
bank 2: account = -135  demanded credit =   0 allowed credit = 0 unpaid =  5
bank 3: account =  -50  demanded credit =   0 allowed credit = 0 unpaid =  5

disposableIncome = 10  desiredConsumption = 20
```

Step 5: credit supply

The bank decides about the amount of loans to give out. Suppose allowed credit is 18

```
bank 1: account =    0  demanded credit = -20 allowed credit = -18 unpaid =  0
bank 2: account = -135  demanded credit =   0 allowed credit =   0 unpaid =  5
bank 3: account =  -50  demanded credit =   0 allowed credit =   0 unpaid =  5

disposableIncome = 10  desiredConsumption = 20
```

Note that the funds now available to the consumers are
`disposableIncome + allowed credit=10+18=28`

step 6: adjust desired consumption according to allowed credit

In this step, the allowed consumption is introduced and it is given by the difference between the demanded and the allowed credit. In any case, the allowed consumption cannot be lower than the subsistence level.

In the example we have

```
bank 1: account =    0  demanded credit = -20 allowed credit = -18 unpaid =  0
bank 2: account = -135  demanded credit =   0 allowed credit =   0 unpaid =  5
bank 3: account =  -50  demanded credit =   0 allowed credit =   0 unpaid =  5

disposableIncome = 10 desiredConsumption = 20 allowedConsumption = 18
```

Now, the consumer goes in the goods market trying to satisfy her desired-allowed consumption. It may happen that goods are in short supply. Suppose this is the case and she can buy 15 instead of 18.

step 7: the consumer adjust bank accounts

The consumer can thus consume 15. 10 of them are payed with disposable income and 5 of them are borrowed from bank 1.

Bank 1 is also willing to lend additional resources (18-5=13), so the consumer uses them to satisfy bank 2 and 3 refunding requests:

```
bank 1: account =  -15  demanded credit = -20 allowed credit = -18 unpaid =  0
bank 2: account = -130  demanded credit =   0 allowed credit =   0 unpaid =  0
bank 3: account =  -45  demanded credit =   0 allowed credit =   0 unpaid =  0

disposableIncome = 0  desiredConsumption = 20 allowedConsumption = 18
                                              effectiveConsumption = 15
```

4.2.3 The process that leads to production

The bulk of this process consists in firms' attempt to adjust inputs to be able to make the production needed to satisfy the demand expected for the next period.

First of all, firms have to decide the level of production to be realized in the next period. To this aim, they have two important signals from the present period: the level of desired demand from consumers and that from other firms (the latter relates to investment as we will clarify below). Concerning the demand from consumers, we recall that the quantity sold by a firm can be different from that initially demanded by consumers. Indeed, the discussion in the previous section points out that when a consumer does not find enough goods of a given type, her/his demand is moved to other goods. However, the firm is informed on the initial level of demand (that desired by consumers) on the item it produces. It is thus natural a firm would realize a production equal to the demand initially claimed by consumers.

Inputs already available to the firm can be higher or lower than those needed to realize the target production. Inputs excesses are someway easier to manage than shortages: workers can be fired while a part of the available production capital can stay unproductive. The upward adjustment is instead more tricky because several impediments to reach the desired level of inputs can occur. The level of financial resources, including new available credit, can be the first constraint to the inputs increase. The availability of the inputs on the market is a second constraint. Finally, if one of these constraints is binding for one of the inputs, the demand of the others must be adjusted coherently. Inputs shortages deserves additional care by making a distinction between inputs that can be produced and those whose production is not possible or cannot be easily obtained in the short run.

As already mentioned above, in the present version of the model we have two inputs: production capital and labor. We treat production capital as "producible" while labor as non producible. So when a production capital shortage is detected, it can be produced to satisfy the shortage. However, the model checks for the existence of unused production capital before new production capital is asked. Production capital, as labor, preserves (in full or in part) its production power when it is moved to another firm. We account for this by including a market for existing unused production capital. So, firms that needs additional production capital, first try to buy already existing production goods that are not used by other firms. Firms that after this adjustment still needs production capital ask for new production goods to other firms. To keep the model simple, we assume that production goods are made up of the same goods

available to consumers, so that, at the end, new investments affect the demand on the existing goods markets.

The actions leading to production are numerous and not so straightforward. To ease the understanding of their unfolding it is convenient to refer to the violet part of figure 4.3. We also report them hereafter as a list. We provide the reader with a "coarse grain" description with the gray backgrounded items.

- check available financial resources for upward adjustment of production capital. This phases is performed by taking the following steps:

  12. compute the economic result and capital depreciation;
  13. perform firms exit;
  14. banks account interests and ask for loan repayments to indebted firms;
  15. firms refund if possible;
  16. firms step the product innovation process;
  - - the software performs a technical step by resetting some variables of the bank accounts;
  17. firms ask for new credit;
  18. banks decide how much credit to allow;
  19. firms compute investment demand and supply and adjust unpaid amount if possible.
  20. entry of new firms that will ask for production capital.

- Labor force adjustment taking in mind the possible financial constraints

  20. Firms perform labor force downward adjustment;
  21. Unemployeds send curricula;
  22. Perform labor force upward adjustment using decentralized matching;
  23. Perform labor force upward adjustment using centralized matching;

- Production capital adjustment taking in mind the possible workers constraint

  24. Firms try to adjust their production capital on the market for used production capital;
  25. Firms that need additional capital ask for new production goods;

- Consumer turnover

  26. perform consumers turnover

- Make production

27. firms make production

We will now enter into details of each step. The gray backgrounded headings will signal when the detailed description leaves one group to enter the next group of actions.

Check available financial resources

Compute the economic result and capital depreciation

Each firm economic result ($\pi_{f,j}$) is computed as follows

$$\pi_{f,j} = D_{f,j\leftarrow c} + D_{f\leftarrow f,j} - W_{f,j}$$

where $D_{f,j\leftarrow c}$ is demand from consumers, $D_{f\leftarrow f,j}$ is demand from firms and $W_{f,j}$ is the amount of wages payed by the firm.

$W_{f,j}$ is the sum of the individual wage (w_w) payed by the firm to the workers it employs. We saw above how w_w is computed (see equation 4.2), so we can write

$$W_{f,j} = \sum_{w \text{ in } f} w_{w,f,j}$$

Profit (a positive economic result) can be used to buy production capital if needed. To see if profit is to be used to buy production capital, we have to update the state of this variable taking account of its depreciation.

The production capital of the firm is denoted with K_f. We divide this amount into two parts: the production capital that was employed in the production and that which is in excess (if any). Remembering the production function, the production capital used in the production is $Y_{f,j}$ and the unused part (if any) is $K_f - Y_{f,j}$.

In this model, these two parts of production capital have two distinct depreciation rate θ_{drD} and θ_{drU}.

The production capital is thus updated as follows:

$$K_f = Y_{f,j}(1 - \theta_{drD}) + (K_f - Y_{f,j})(1 - \theta_{drU})$$

We have thus defined the following parameters

notation in equations	name in code	value	read from file
θ_{drD}	`Context.percentageOfUsedCapitalDepreciation`	0.01	yes
θ_{drU}	`Context.percentageOfUnusedCapitalDepreciation`	0.0	yes

The UML sequence diagram of this event is available here.

Perform firms exit

In this basic version of the model, firms turnover is very simple: each firm that becomes too small is replaced with a new firm.

In particular, the exit condition is

$$D_{f,j\leftarrow c} + D_{f,j\leftarrow c} < \bar{D}$$

where we use the following parameter

notation in equations	name in code	value	read from file
$\bar{D}$	`Context.thresholdDemandForFirmsExit`	20	yes

In this part of the code, the list of new entrants is created but firms are not yet included in the model. This is because new entrants have to participate in the production factor markets in order to start the production. The demand for production capital that will be computed below will account for the demand of both incumbent and new firms. This will also apply to the labor demand.

The UML sequence diagram of this event is available here.

Banks account interests and ask for loan repayments to indebted firms

The accounting of interest updates the firms bank accounts.

Positive bank accounts are updated using the interest rate on deposits i^+, so that if $BA_{fb} \geq 0$

$$BA_{fb} = BA_{fb}(1 + i^+)$$

Differently from consumers, firms have not a subsidized interest rate, and negative bank accounts are charged by the ordinary interest rate (i^-):

$$BA_{fb} = BA_{fb}(1 + i^-)$$

Once the accrued interest has been accounted, the bank may ask the reduction of some negative accounts. The amount of a negative bank account owned by a firm desired by the bank is thus determined as follows:

$$BA^{db} = \begin{cases} BA_{fb} & \text{with probability } pr_{fbren} \\ BA_{fb}\theta_{fbncr} & \text{with probability } 1 - pr_{fbren} \end{cases}$$

The parameters involved in this computation are:

notation in equations	name in code	value	read from file
θ_{fbncr}	`Context.percentageOfOutstandingCreditAllowedTo FirmsWhenCreditIsNotCompletelyRenewed`	0.9	yes
pr_{fbren}	`Context.firmsProbabilityToHaveOutstanding DebtCompletelyRenewed`	0.5	yes

The UML sequence diagram of this event is available here

Firms refund if possible

In the previous step banks can ask firms to reduce amount of negative bank accounts.

In this model firms can have more than one bank account. In principle, some of them can be positive, and others negative. The downward adjustment can be asked on some of the negative accounts. Firms can face banks refunding requests using all their financial assets: cash flow and their positive bank account.

In case refund is asked on some accounts, the entrepreneur starts checking its bank account list from the beginning and when s/he find a positive amount use it to fulfill the other banks refunding requests. If the positive amounts are not enough, the cash on hand is used. If Even cash on hand is not enough, a positive unpaid amount is recorded.

The UML sequence diagram of this event is available here

Firms step the product innovation process

Innovation in this model comes into the form of product innovations. We model this process by introducing the variable AR_f which denotes the Absolute Rank of the good produced by firm f. AR_f gives the position of the good in the quality ladder where higher values denote higher quality.

The product innovation is implemented in a very simple way

$$AR_f = \begin{cases} AR_f + 1 & \text{with probability} \quad pr_{pi} \\ AR_f & \text{with probability} \quad 1 - pr_{pi} \end{cases}$$

The parameters involved in this computation is:

notation in equations	name in code	value	read from file
pr_{pi}	`Context.probabilityOfAProductInnovation`	0.0	yes

The UML sequence diagram of this event is available here

We saw above that the office for statistics allocate demand. In doing that it accounts for goods quality: higher quality attracts more demand. This implies that firms that do not innovate are penalized. This can lead less advanced products to disappear.

As time passes, a $pr_{pi} > 0$ implies that absolute ranks of the products that are exchanged in the economy always increase. Because low ranked goods disappear, we introduce the Relative Rank RR_f to perform computation. It is defined as:

$$RR_f = AR_f - \min(AR_f) + 1$$

In this way, even if the absolute rank of the less advanced good that is exchanged at a given time ($\min(AR_f)$) increases, its relative rank is alway 1.

Consider for example companies producing smart phones. Suppose we identify these companies with uppercase letters A, B, C and so on. Suppose furthermore that they name their product by the company name and the absolute rank of the product. Suppose the variety of smart phones available on the market is A6, B6, C5. We can compute the relative rank as explained above. The following table reports the situation

f	product	AR_f	RR_f
A	A6	6	2
B	B6	6	2
C	C5	5	1

Suppose company A launches its new A7 smart phone. The previous table evolve as follows

f	product	AR_f	RR_f
A	A7	7	3
B	B6	6	2
C	C5	5	1

After the introduction of this new product, consumers see C5 outdated. So company C either exits from the market or succeeds in innovating and launches the C6. In both cases, the relative rank is updated:

if C exits

f	product	AR_f	RR_f
A	A7	7	2
B	B6	6	1

if C innovates

f	product	AR_f	RR_f
A	A7	7	2
B	B6	6	1
C	C6	6	1

Firms ask for new credit

Firms may need new credit to increase the level of their production capital or to pay residual unpaid amounts on banks accounts.

First, a firm evaluates if it has to increase its production capital. In doing that it has to forecast the level of next period demand. We know there are two sources of demand: from consumers and from firms. To forecast the consumers demand, the firm uses the desired-allowed demand $D^a_{f,j\leftarrow c}$ as computed in equation 4.3. The demand from other firms is assumed to be equal to that received in the previous period. The demand expected by the firm is thus

$$D^e_f = D^a_{f,j\leftarrow c} + D_{f,j\leftarrow f}$$

Now, remembering the production function, we have that the desired level of production capital is:

$$K^d_f = D^e_f$$

If

$$K^d_f > K_f$$

financial resources are needed to make the desired adjustment. The firm checks its internal financial resources, i.e. cash on hand and positive bank accounts and, if they are not enough, asks for credit.

If existing production capital is enough to produce the expected demand, credit could be also needed; this happen if unpaid amounts exist.

If credit is needed, the entrepreneur checks the possibility to ask it to the banks it is customer of. This possibility is prevented if positive unpaid amount are present in all the firm's bank accounts. If one or more bank accounts have no unpaid amount, the entrepreneur choses the bank account having the best account to ask new credit. When searching for the best bank account, the code also identifies the worst bank; it will be used to deposit the residual cash on hand that possibly remains at the end of the adjustment process.

The UML sequence diagram of this event is available here

Banks decide how much credit to allow

In the previous step, firms who need credit set the desired amount on one (the best) of their bank accounts having null unpaid amount if such account exists. Let us identify this variable with BA_f^d (where the d upper script means **d**esired). Because they are asking for credit, this amount is negative: $BA_f^d < 0$. Note that the best bank account con have either a positive or negative amount.

In the present version of the model, the bank decides the allowed credit $BA_f^{ab} < 0$ (**a**llowed by **b**ank to **f**irm) as follows

- if $BA_{fb} \geq 0$ and $BA_f^{ab} < 0$

$$BA_f^{ab} = \begin{cases} BA_f^d & \text{with probability } pr_{abf} \\ \theta_{abf} BA_f^d & \text{with probability } 1 - pr_{abf} \end{cases}$$

- if $BA_{fb} < 0$ and $BA_f^{ab} < 0$

$$BA_f^{ab} = \begin{cases} BA_f^d & \text{with probability } pr_{abf} \\ BA_{fb} - \theta_{abf}(BA_{fb} - BA_f^d) & \text{with probability } 1 - pr_{abf} \end{cases}$$

We have thus the following parameters:

notation in equations	name in code	value	read from file
θ_{abf}	`Context.percentageOfNewDemandedCredit AllowedToFirmsWhenCreditIsNot CompletelyAllowed`	0.9	yes
pr_{abf}	`Context.firmsProbabilityToHaveNewDemanded CreditCompletelyAllowed`	0.5	yes

The UML sequence diagram of this event is available here

Firms compute investment demand and investment supply; they adjust unpaid amount if possible

The following variables are managed in this step

- `firmInvestment`,
- `cashOnHand`,
- `promissoryNotes`,
- `unpaidAmount`

The way they are managed depends on the outcome of the previous steps. Because there are a variety of possible states a firm can be at this stage, we will give here a selection of examples starting from bad to good situations.

In the worst case,

- the firm suffered a loss;
- available financial resources are not enough to cover it;
- a bank that lend money cannot be found;
- one or more banks ask for refunding.

In this case we allow the firm to issue promissory notes that are delivered to consumers. These promissory notes will be payed in the future whenever possible. It is highly probable that the loss is caused by a fall in demand, so that the firm want to decrease its production capital. The variables listed above will probably be as follows:

- `firmInvestment` < 0,
- `cashOnHand` $= 0$,
- `promissoryNotes` > 0,
- `unpaidAmount` > 0

In another difficult situation the firm has profits and positive bank accounts that are not enough to pay back the bank. In this case the firm end up with positive unpaid amounts:

- `firmInvestment` < 0,
- `cashOnHand` $= 0$,
- `promissoryNotes` $= 0$,
- `unpaidAmount` > 0

A better situation is the one in which the fall of demand is not so heavy to cause a loss:

- `firmInvestment` < 0,
- `cashOnHand` > 0,
- `promissoryNotes` $= 0$,
- `unpaidAmount` $= 0$

A promising situation is that in which the level of demand increase so that the firm want to invest and it has a positive economic result. Here we can distinguish between the case in which the cash on hand is not enough to finance investments:

- `firmInvestment`>`cashOnHand+sum of positive banck accounts` > 0,
- `promissoryNotes` $= 0$,
- `unpaidAmount` $= 0$

and the case in which the cash on hand is higher than investments:

- `firmInvestment`<`cashOnHand+sum of positive banck accounts` > 0,
- `promissoryNotes` $= 0$,
- `unpaidAmount` $= 0$

In the latter case the cash on hand in excess is deposited in the worst bank account.

Even though the variety of cases presented is incomplete, it can be used to star reasoning on the supply and demand of production capital.

It is straightforward that entrepreneurs who want to disinvest, i.e. those having `firmInvestment` < 0 are offering their production capital excesses in the used production capital market. However, as we will see in one of the following step, the sum of these desired disinvestments cannot be considered as the supply in the existing production capital market. This is because we have to take into account the possibility of existing production goods to be used by other firms. In other words we have to account for the degree of investment reversibility.

The demand of new production capital is affected by the credit supply. At the current state, some firm asked for new credit while other do not. Those which asked for, now know the amount of credit the best bank (if it exists) is willing to allow. Therefore, they can check if the available credit is enough to cover their losses or to realize the desired investments. Firms that want to

invest, both those who asked for credit and those who do not, now know the level of investments they could achieve according to the available financial resources. However, even though the firm has enough financial resources we cannot set the amount of investment demand at this stage: firms have first to check for the availability of workers. If the needed workers will not be found, the desired level of investment will be resized. Therefore, the demand of investments will be set in a later step.

The UML sequence diagram of this event is available here

Entry of new firms that will ask for production capital

In the previous step we computed the potential demand and supply of production capital by incumbent firms. However, the demand of production capital must be integrated with that of firms that intend to enter the market. The entering of new firms also impacts on banks because they are involved in the financing of these new activities. Furthermore, the new entries modifies the position each firms has on the goods market because it will attract a share of demand.

All these aspects are managed in this step. In particular, the desired size of each new firm is a parameter:

notation in equations	name in code	value	read from file
K_{entry}	`Context.productionOfNewEnteringFirm`	50	yes

This amount is also identified as `firmInvestment` and thus enters in the demand for production capital.

New firms thus opens a number of bank accounts and ask for funds to finance the new investment to one of these banks (the fist one created for coding convenience). The bank allows for all the credit asked by a new firm.

Like incumbent firms, the production capital of new entrants will be resized in case of a work force shortage.

The UML sequence diagram of this event is available here

Labor force adjustment

Firms perform labor force downward adjustment

The downward adjustment of the labor force has two motivations:

- workers retirement;
- excess of labor production capacity with respect to the expected demand.

Concerning retirement, the software checks each worker's age against the parameter:

notation in equations	name in code	value	read from file
θ_{retire}	`Context.ConsumerExitAge`	70	yes

The check is made by each firm, and if the worker's age is higher than the retirement age, the worker is removed from the workers list.

Once, retirement is performed, the firm computes its workers production capacity. If it is higher than the demand expected for the next period, the firm fire workers starting from the bottom of its workers list i.e. adopting a last in, first out method. The firm continue firing until firing an additional worker will bring the workers production capacity lower than the expected demand.

The UML sequence diagram of this event is available here

Unemployeds send curricula

The way Unemployed people send curricula depends on the labor market matching mechanism. The latter can be chosen by setting the following parameter:

notation in equations	name in code	value	read from file
$\theta_{LMmatch}$	`Context.FirmsWorkersMatching`	0	yes

It can be set as follows:
0 perform a decentralized mechanism only;
1 perform first a decentralized matching followed by a centralized allocation of residual vacancies;
2 perform a centralized matching only.

If the decentralized matching mechanism is involved (cases 0 and 1), unemployed consumers are allowed to select a number of firms given by the parameter:

notation in equations	name in code	value	read from file
θ_{njasu}	`Context.numberOfJobApplicationAnUnemployedSends`	2	yes

and send to each of them the curriculum.

If the centralized matching mechanism is involved (cases 1 and 2), unemployed consumers send their curriculum to the labor office.

So, in case 0, unemployed consumers send θ_{njasu} curricula, in case 1 they send θ_{njasu} +1 curricula while in case 2 they send just one curriculum.

The UML sequence diagram of this event is available here

Perform labor force upward adjustment

In this step, firms that want to increase their production try to hire new workers. At this stage they know the level of production capital that can be achieved given the credit allowed by banks. The attempt here is to line up the workers production capacity to the capital production capacity allowed by banks.

Workers search mechanisms depends on the matching mechanism chosen (the $\theta_{LMmatch}$ parameter presented in the previous paragraph). If the decentralized matching mechanism is involved (cases 0 and 1), each firm hires among those who sent the curriculum to it (those who were already hired by other firms are not considered by the firm).

In case 1, the residual vacancies are posted to the labor office, while in case 2, all the vacancies are posted. The Labor office will try to fill the vacancies.

The UML sequence diagram of this event is available here

Production capital adjustment

Firms try to adjust their production capital on the market for used production capital

As explained above, it can happen that workers production capacity and potential production capital cannot be lined up because of workers shortage. In this case, the capital production capacity, and hence investment demand, is resized to meet the workers production capacity.

After this resizing, the demand of production capital can be finally computed.

The UML sequence diagram of this event is available here

Next, the code checks for possible adjustments on the market for existing unused production capital.

First of all, there is a parameter that transforms unused production capital into reusable production capital.

notation in equations	name in code	value	read from file
θ_{prupc}	`Context.percentageOfRealized UnusedProductionCapital`	0.9	yes

The sequence of events to adjust the production capital is as follows

- computation of aggregate investments and aggregate unused capital;
- unused capital is transformed in reusable production capital;

- reusable production capital is used to satisfy aggregate investments;
- firms that sold unused capital reduce their available production capital and increase their worst bank account;

The UML sequence diagram of this event is available here.

Firms that need additional capital ask for new production goods

If the demand for investments cannot be completely fulfilled by the reusable production capital, firms ask for newly produced investment goods. We denote this variable with I_f. Summing over f we get the aggregate demand of new production goods

$$I = \sum_f I_f$$

The office for statistics is charged for allocating such source of demand. Because it also has the role of allocating the aggregate demand from consumers, the easiest way to proceed is that of using the same allocation method. From this process, each firm will know the demand from other firms, $D_{f \leftarrow f,j}$, that will be added to that from consumers.

The UML sequence diagram of this event is available here

Consumer turnover

Consumer turnover is managed in a very simple way. When a consumer reaches the maximum age θ_{cea} s/he is replaced with a new one having age equal to zero. The new consumers inherit the bank accounts of the exited consumer.

The UML sequence diagram of this event is available here

Make production

Now that firms have performed the possible adjustments on the production capital and labor market, they are ready to perform production.

After the adjustments each firm knows its available production capital (K_f) and the set of productivities of the workers it hired ($\psi_{w,f}$).

This allows the computation of both the production capital potential production and the labor force potential production following the equations given in the initialization section (page 35):

$$Y_f^{PK} = K_f \qquad \text{and} \qquad Y_f^{PL} = \theta_{YL} \sum_w \psi_{w,f}$$

Finally, the firm output is computed using the Leontief type production function (equation 4.1):

$$Y_f = \min(Y^{PK}, Y^{PL})$$

Now, we know that the demand for investment goods $D_{f,j\leftarrow f}$ is satisfied in full before that of consumers. We can think investments products are ordered, and the market for these goods always cleans: $D_{f,j\leftarrow f} = Y_{f,j\rightarrow f}$. This allows us to compute the supply of consumption goods:

$$Y_{f,j\rightarrow c} = Y_{f,j} - Y_{f,j\rightarrow f}$$

The UML sequence diagram of this event is available here

Now we are ready to go again through the process that leads to consumption. However, as mentioned above, new entry of firms changes the market power of incumbents. Thus, a step done before the aggregate demand is allocated is the update of such market shares.

The UML sequence diagram of this event is available here.

4.2.4 Examples of firm-bank relationship

A numerical example of the sequence of actions for a firm in **bad** economic conditions is given hereafter.

Bad condition

Step 1: interests and ask for repayments

Firms can be customers of several banks.

First of all, the banks account the interest rate.

Suppose that after accounting interests we have the following situation

```
bank 1 account =   10
bank 2 account = -150
bank 3 account =  -50
```

The bank assumes that indebted firms ask for the whole renewal of the debt:

```
bank 1 account =   10  demanded credit =    0
bank 2 account = -150  demanded credit = -150
bank 3 account =  -50  demanded credit =  -50
```

Each bank with a negative account can ask for refunding. In this case the allowed credit is lower (in absolute value) to the demanded credit. Suppose bank 2 asks for refunding and bank 3 does not:

```
bank 1 account =   10  demanded credit =    0 allowed credit =    0
bank 2 account = -150  demanded credit = -150 allowed credit = -130
bank 3 account =  -50  demanded credit =  -50 allowed credit =  -50
```

In this example, the firm needs 20 to satisfy banks requests.

Step 2: refunding

The possibility to refund depends on the resources available on banks and on the economic result. In our example, 10 is available in bank 1.

To go on with our example, let us assume that the economic result is `-50` i.e. the firm is suffering a loss.

The firm use 10 available in bank 1, but it is not enough to satisfy banks requests. Shortages are recorded as unpaid amounts.

The firm financial situation is represented as follows

```
bank 1: account =    0  demanded credit =    0 allowed credit =    0 unpaid =  0
bank 2: account = -140  demanded credit = -150 allowed credit = -130 unpaid = 10
bank 3: account =  -50  demanded credit =  -50 allowed credit =  -50 unpaid =  0

cashOnHand = -50
```

Step 3: account resetting

In this step, banks set the demanded and allowed credit to zero.

The new situation is

```
bank 1: account =    0  demanded credit =  0 allowed credit = 0 unpaid =  0
bank 2: account = -140  demanded credit =  0 allowed credit = 0 unpaid = 10
bank 3: account =  -50  demanded credit =  0 allowed credit = 0 unpaid =  0

cashOnHand = -50
```

Step 4: set desired credit

Now the firm can ask for new credit. This can be done for two reasons: 1) to finance new investments and 2) to pay unsatisfied lenders.

Suppose now that our firm does not invest, and asks for credit to pay unsatisfied lenders.

Credit in this step is asked to one of the banks, in particular to that with the "best" account.

In this example, the new asked credit is `10+50=60`. The update situation is

```
bank 1: account =    0  demanded credit = -60 allowed credit = 0 unpaid =  0
bank 2: account = -140  demanded credit =   0 allowed credit = 0 unpaid = 10
bank 3: account =  -50  demanded credit =   0 allowed credit = 0 unpaid =  0

cashOnHand = -50
```

Step 5: credit supply

The bank now decides the allowed credit.

The situation evolves differently according to the allowed amount.

We will present here two situations.

In the firs one, the bank allows all the demanded credit. In this case the accounting evolves as follows

```
bank 1: account =    0  demanded credit = -60 allowed credit = -60 unpaid =  0
bank 2: account = -140  demanded credit =   0 allowed credit =   0 unpaid = 10
bank 3: account =  -50  demanded credit =   0 allowed credit =   0 unpaid =  0

cashOnHand = -50
```

In the second situation, the bank allows only 30 of the 60 that was asked. In this case the accounting is

```
bank 1: account =    0  demanded credit = -60 allowed credit = -30 unpaid =  0
bank 2: account = -140  demanded credit =   0 allowed credit =   0 unpaid = 10
bank 3: account =  -50  demanded credit =   0 allowed credit =   0 unpaid =  0

cashOnHand = -50
```

Step 6: the firm adjust bank accounts

The resources made available by bank 1 are used and the situation evolves as follows.

If the `allowed credit = -60`, the new credit is enough to pay the unpaid amount to bank 2 and to cover the loss:

```
bank 1: account =  -60  demanded credit = -60 allowed credit = -60 unpaid =  0
bank 2: account = -130  demanded credit =   0 allowed credit =   0 unpaid =  0
bank 3: account =  -50  demanded credit =   0 allowed credit =   0 unpaid =  0

cashOnHand = 0
```

If the `allowed credit = -30`, new credit is not enough. The entrepreneur now uses the 30 to partially cover the loss. The residual loss is covered by issuing promissory notes. The unpaid amount remains in bank 2:

```
bank 1: account =  -30  demanded credit = -60 allowed credit = -30 unpaid =  0
bank 2: account = -140  demanded credit =   0 allowed credit =   0 unpaid =  10
bank 3: account =  -50  demanded credit =   0 allowed credit =   0 unpaid =  0

cashOnHand = 0 promissoryNotes=20
```

A numerical example of the sequence of action for a firm in **good** economic conditions is given hereafter.

Good conditions

Step 1: interests and repayments

Suppose we start from the same conditions as in the bad case.

There are no differences in the evolution of this step, so that the situation at the end of this step is the same:

```
bank 1 account =   10  demanded credit =    0 allowed credit =    0
bank 2 account = -150  demanded credit = -150 allowed credit = -130
bank 3 account =  -50  demanded credit =  -50 allowed credit =  -50
```

Step 2: refunding

Good conditions here means that the firm realized a profit and thus has a positive cash on hand, say

`cashOnHand = 50`

20 of them are used to refund the bank. So, now, the situation is

```
bank 1: account =   10  demanded credit =    0 allowed credit =    0 unpaid =  0
bank 2: account = -130  demanded credit = -150 allowed credit = -130 unpaid =  0
bank 3: account =  -50  demanded credit =  -50 allowed credit =  -50 unpaid =  0

cashOnHand = 30
```

Step 3: account resetting

```
bank 1: account =   10  demanded credit =    0 allowed credit =    0 unpaid =  0
bank 2: account = -130  demanded credit =    0 allowed credit =    0 unpaid =  0
bank 3: account =  -50  demanded credit =    0 allowed credit =    0 unpaid =  0

cashOnHand = 30
```

Step 4: set desired credit

Imagine now, the firm had a production capital equal to 100 before starting production.

Suppose production depreciates capital to 95.

Furthermore, suppose the firm expects an increase of demand and wants to increase its production capital to 110.

So, to bring production capital to the desired level 15 is needed.

Checking financial resources available internally the firm conclude that it can achieve the objective without asking to banks.

Furthermore there is an inconsistency in firms bank accounts: the positive one should be used to reduce the negative ones. The software at this stage withdraws positive bank accounts and store them in the variable `financialResourcesInBankAccounts`:

```
bank 1: account =    0  demanded credit =    0 allowed credit =    0 unpaid =  0
bank 2: account = -130  demanded credit =    0 allowed credit =    0 unpaid =  0
bank 3: account =  -50  demanded credit =    0 allowed credit =    0 unpaid =  0

cashOnHand = 30 financialResourcesInBankAccounts = 10
```

Step 5: credit supply

In this step no change is performed because credit was not asked.

Step 6: adjust production capital and banks accounts

Production capital is adjusted by using internal financial resources.

So, after this step we have

`productionCapital=110`

`cashOnHand + financialResourcesInBankAccounts = 25`

finally, the residual internal funds are used to improve the "worst" bank account:

```
bank 1: account =    0  demanded credit =    0 allowed credit =    0 unpaid =  0
bank 2: account = -105  demanded credit =    0 allowed credit =    0 unpaid =  0
bank 3: account =  -50  demanded credit =    0 allowed credit =    0 unpaid =  0

cashOnHand = 0 financialResourcesInBankAccounts = 0
```

Chapter 5

Selected topics

5.1 Stock-Flow consistency

Uncontrolled flows

Part III

Modifying GABRIELE

Chapter 6

How to modify GABRIELE

6.1 Initialization

The step for initialization are performed in the `Context.java` file.
In particulate, the lines of code between the
`System.out.println("BEGINNING OF INITIAL SETUP");`
and the
`System.out.println("END OF INITIAL SETUP");`
instructions perform the initialization steps.

6.2 Main loop

The main loop is scheduled in the
`OfficeForStatistics.scheduleEvents()`
method.

Part IV

Documenting GABRIELE

Chapter 7

LaTeXdocumentation

The LaTeXtypesetting system is used to produce the GABRIELE manual.

All the files needed to obtain the electronic version of the manual are in the `docs/latexdoc` folder.

The manual source file is named `manual.tex`

In the following subsection we report on the various tools used to produce the additional elements that are integrated into the main file.

7.1 Additional lists in the table of contents

The `glossaries` package is used to produce the additional lists to be included in the table of contents.

Here we will briefly explain how it works. First of all a new glossary must be created. In the following example, the glossary called `variable` is created:

```
\usepackage[toc]{glossaries}
\newglossary[nlg]{variable}{vin}{vot}{List of variables}
```

The glossary entries are defined in a separate file that will be included in the main document. Suppose the entries for the `variable` are created in the `glossary_notation_variables.tex` file.

The entries definitions are imported in the preamble with the following command

```
\input{glossary_notation_variables}
```

Glossaries entries definition format is as follows:

```
\newglossaryentry{var:WWf}% label
{
  type=variable,% glossary type
  name={$WW_{f,j}$},
  description={Sum of wages payed by firm $f$ producing good $j$}
}
```

Note that the type entry must report the glossary name.

Once the glossary entry is defined in the `glossary_notation_variables.tex` file, it can be used in the main document. To do that, the `\gls{glossary entry label}` must be used. In our case, for example we can write:

```
where \gls{var:WWf} denotes the sum of wages ...
```

Finally, the `\printglossaries` command must be included in the main file to make the defined glossaries show up.

An additional compilation step is needed to generate the glossaries.

Once compiled a first time using the `pdflatex` command, the `manual.ist` file is created. Now give the command

```
makeindex -s manual.ist -o manual.vin manual.vot
```

to generate the auxiliary files taken as input by the following `pdflatex` runs. Note that we defined the extension of the files used in the `makeindex` command in the `\newglossary` line options.

Presently, we have created two glossaries, one for variables and one for parameters. To simplify the updating of the auxiliary files, we create the `makeglossaries` script. So, to update the master document when a new glossary entry is created and cited, enter the following commands in the command line:

```
pdflatex
./makeglossaries
pdflatex
pdflatex
```

7.2 Figures

Some figures (for example 3.1-3.4) are obtained by using metapost. The metapost work flow is as follows: metapost code is written in an ascii file. Once finished, the file is compiled with the `mptopdf` command. If the command is not in the system it can be installed. Alternatively, the final pdf can be obtained by fist compiling with the `mpost` command and then converting the output using `epstopdf`. Therefore, to modify these figures, you have to edit the corresponding mp source file an recompile it.

Figures 4.1 and 4.3 are also obtained as metapost output, but the process is more structured to ease possible changes. The complication of this figure is that events are represented as evenly spaced sun rays. Imagine a new event must be added. It would be a significant work to manually change the positions of all the events labels. We use the R software (https://www.r-project.org) to "automate" the process that leads to the figure. The `visual.R` script, for example, produces figure 4.1. It takes as input two text files: `visual.txt` and `items_in_main_loop_black.txt`. The first one contains metapost drawing commands for all the elements except the events descriptions (the sun rays). This file has to be edited to include new such elements. The second file lists the labels of the rays (note, they are in inverted order and include LATEXcommands).

The R script loads the list from the file, computes the angle between rays and produces metapost command to draw the labels. All the metapost drawing commands are written in the `visual.mp` file. The R script also takes care of running metapost on this file. At the end of the day, to modify the figure one has to edit one of the text files and run the script in R. The same process applies to the production of figure 4.3. The file involved here are `visual1.txt` and `items_in_main_loop.txt` and the R script `visual1.R`.

Chapter 8

Javadoc

From Repast project menu, choose Generate javadoc. Select gabriele from the list. Modify the destination folder to `<path to repast workspace>/gabriele/docs` if needed and click on finish.

Chapter 9

Unified Modeling Language (UML)

9.1 Class diagrams

Class diagrams can be generated directly from the code by using the umlgraph.[1] As trated in the umlgraph installation page: UmlGraph needs to be post-processed with the Graphviz dot program. Therefore, to draw class diagrams with UmlGraph class you will need to have javadoc and Graphviz[2] installed on your computer.

First install the graphviz software. Now you should have the `dot` command available in your system.

Now download the umlgraph and note down the path to its jar file.

There are several ways to use the umlgraph program to generate class diagrams. Here we will give hints on how to use it via javadoc.

Documentig a Class

To document a class, the `UmlGraph` docled have to be used. It is supplied as the `org.umlgraph.doclet.UmlGraph` class in the umlgraph jar.

The command to be given is:

```
javadoc -d <outputfolder>
-docletphath <path to umlgraph jar file>
-doclet org.umlgraph.doclet.UmlGraph <options> <javafile with complete phath>
```

See the umlgraph documentation for options.[3] As a first hint, the `-all` option is perhaps the best one in case you are documenting a single class.

[1] https://www.spinellis.gr/umlgraph/
[2] http://www.graphviz.org
[3] https://www.spinellis.gr/umlgraph/doc/cd-opt.html

The command will generate a `graph.dot` file inside the `outputfolder` you specified with the `-d` option.

Now, you have to run the `dot` on the `graph.dot` file to obtain a picture of the class diagram.

`dot -T<outputformat> -o<outputfilename> graph.dot`

As an example,

`dot -Tjpg -otmp.jpg graph.dot`

produces the `tmp.jpg` file.

Documentig packages

To document classes in packages, the `UmlGraph` docled have to be used as above.

The command to be given is:

```
javadoc -d <outputfolder>
-sourcepath <path to the packages source files>
-docletphath <path to umlgraph jar file>
-doclet org.umlgraph.doclet.UmlGraph <options> <packages separed by a white space>
```

See the umlgraph documentation for options. As a first hint, the `-all` option should be avoided while the `inferrel` should be used to highlight the relationships among classes.

As above, the command will generate a `graph.dot` file inside the `outputfolder` you specified with the `-d` option.

Post process the file with the `dot` program as explained above to obtain a picture.

API documentation with UML class diagrams

9.2 Sequence diagrams

Use umlet