GABRIELE

General Agent Based
Repast Implemented Extensible
Laboratory for Economics

Users Manual

Gianfranco Giulioni

July 14, 2016

To Gabriele

Contents

List of variables

$CoH_{f,j}$ Cash on hand of firm f producing good j. 27

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

$D_{f\leftarrow h,j}$ Demand of consumption goods. 27

$WW_{f,j}$ Sum of wages payed by firm f producing good j. 27

List of Parameters

$\phi_{f \leftarrow h,j}$ share of something. 27

Part I

Setting up GABRIELE

In this part of the manual we describe two ways of setting up for running the model. The first one is the standard way where one can take advantage of various wizards to set up and running the model. Although this provides several facilities, some user could feel uncomfortable with this standard process and would like to revert to a more basic process. The streamlined installation is for these users. It behooves to point out that the streamlined process was especially designed for running the model in the local machine and in batch mode. Therefore, funs of the Unix like command line interface will particularly enjoy 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 the model. 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 in the list of installed software if JDK is installed in your system. If yes, note the JDK version.

The fastest way to check all this is to 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 check the leatest released version of JDK.

If JDK is not installed in your system or 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 completed 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. If not, it must be added manually. 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 the eclipse downloaded and installed as described in the previous section.

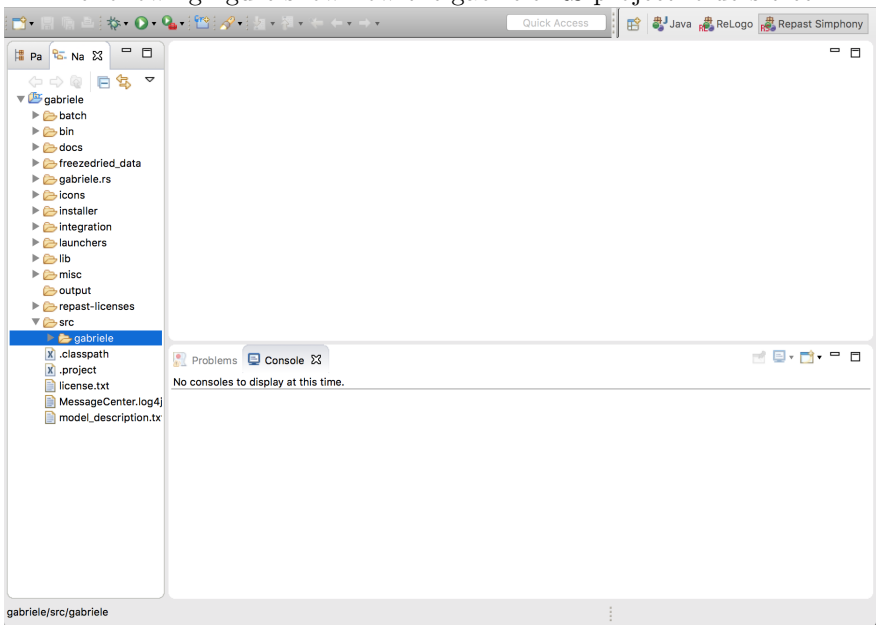
Suppose your workspace has the following path:
`/Users/coolcoder/Documents/workspace`

Open the RS perspective (window → open perspective).

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

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

The following figure show how the gabriele RS 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 the best option to share the code developments you will do.

So, let us show how to fetch the GABRIELE files via git. To do that you must have a git client installed in your system. Many system comes with a git client already installed, otherwise you have to install it. Mac and windows users can consider to install the GitHub Desktop software.

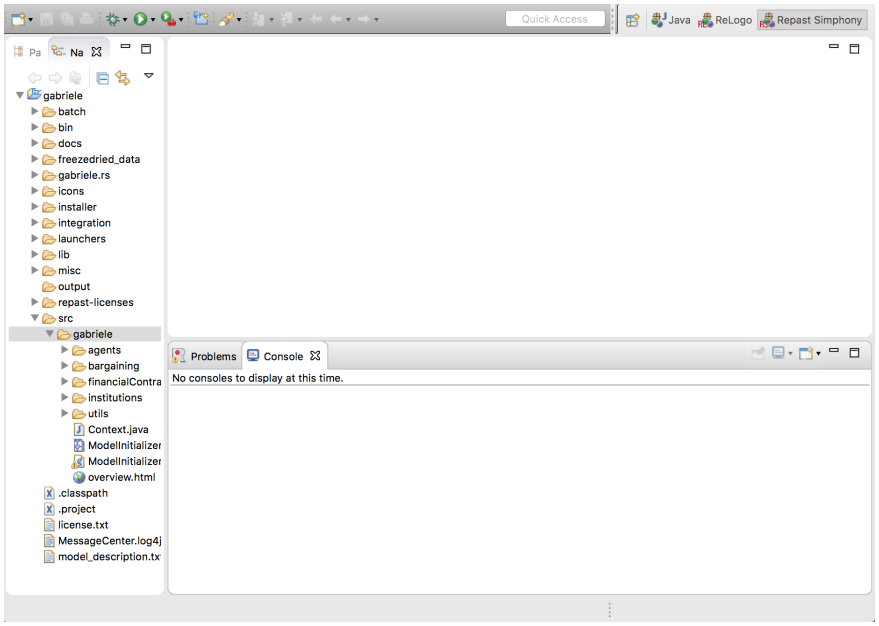
You can verify if git is installed in your system by checking if your command line interface recognize the `git` command. If your check is successful, change directory to the gabriele project folder:
`cd /Users/coolcoder/Documents/workspace/gabriele`
and type the following commands:

```
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 folders should look like.



Using a 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.

Unpacking it, the gabriele-master folder is created. Move the whole content of this folder in the gabriele RS project folder:
`/Users/coolcoder/Documents/workspace/gabriele/`
if your procedure asks about what should be done with existing files and folder, choose to overwrite them (merging the folders).

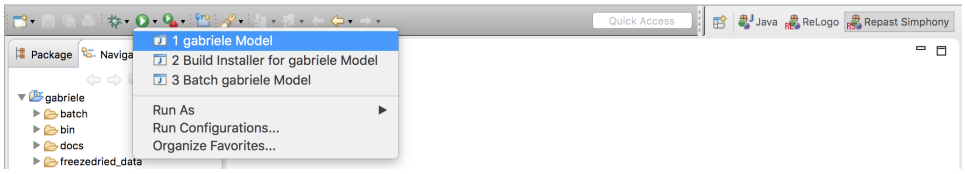
Now refresh eclipse (file → refresh).

1.2 Testing the Installation

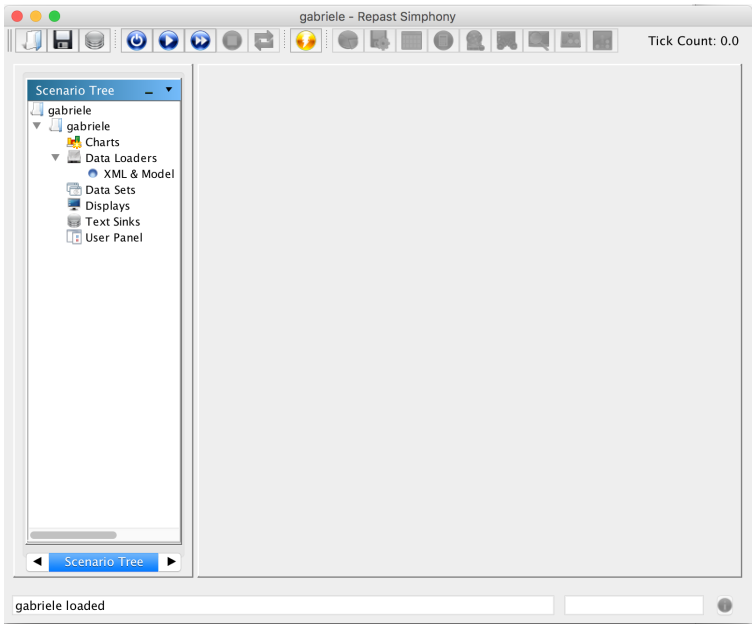
1.2.1 Setup the data loader

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

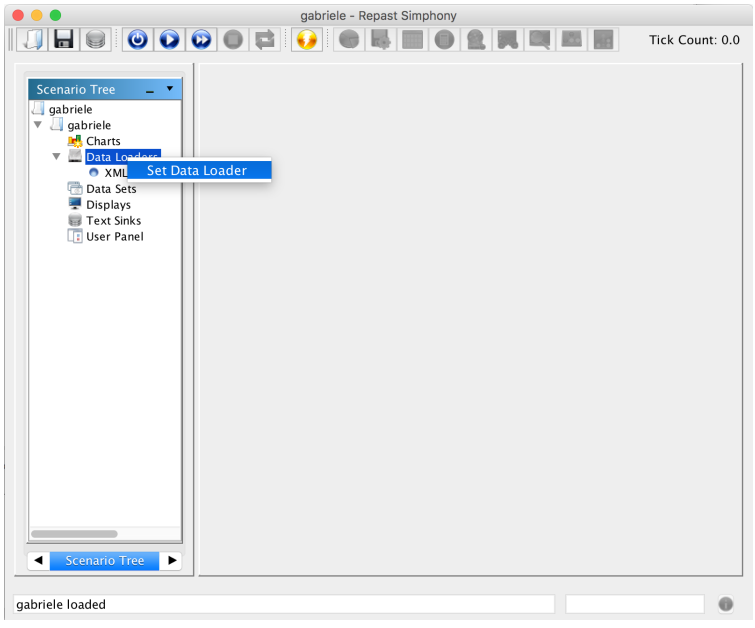
After clicking, a menu opens as shown by the following figure. Click the `gabriele Model` item



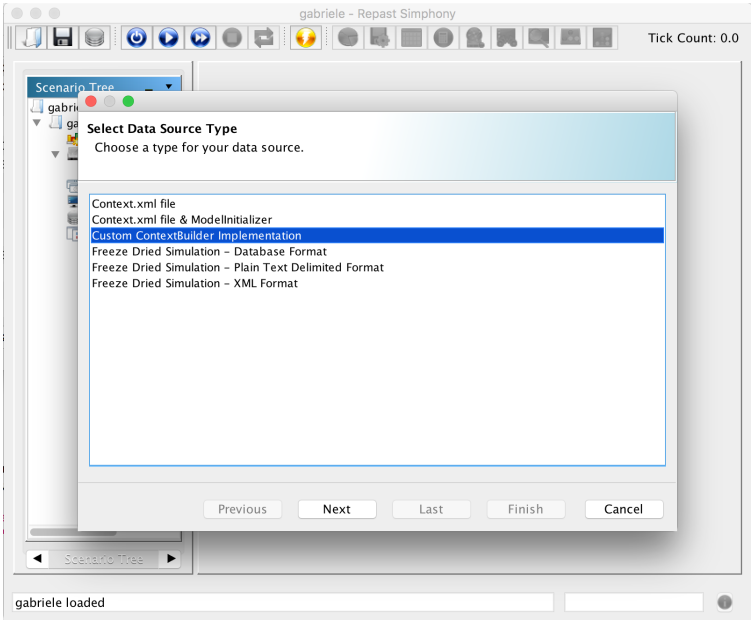
After a while, the RS GUI displayed in the following figure will show up



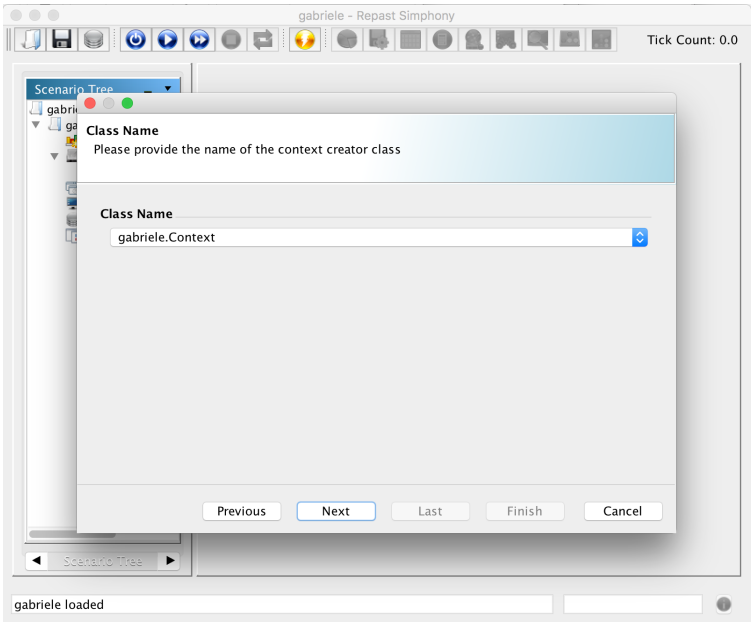
The first time you start the RS GUI with the GABRIELE scenario you have to setup the custom dataloader. As you can see in the scenario tree under the Data Loades item the XML & Model data loader is present. This must be changed in the custom loader for this model. To do so right click on the Data loaders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



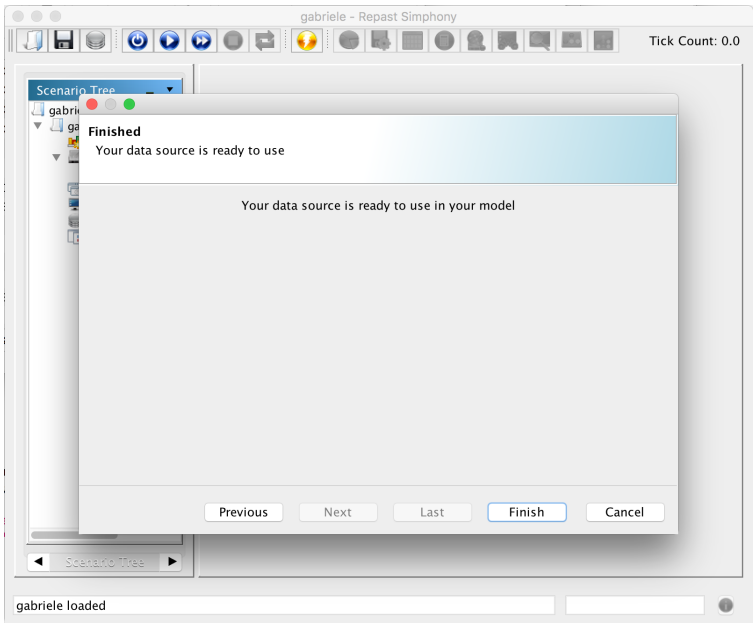
The following new window will appear



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

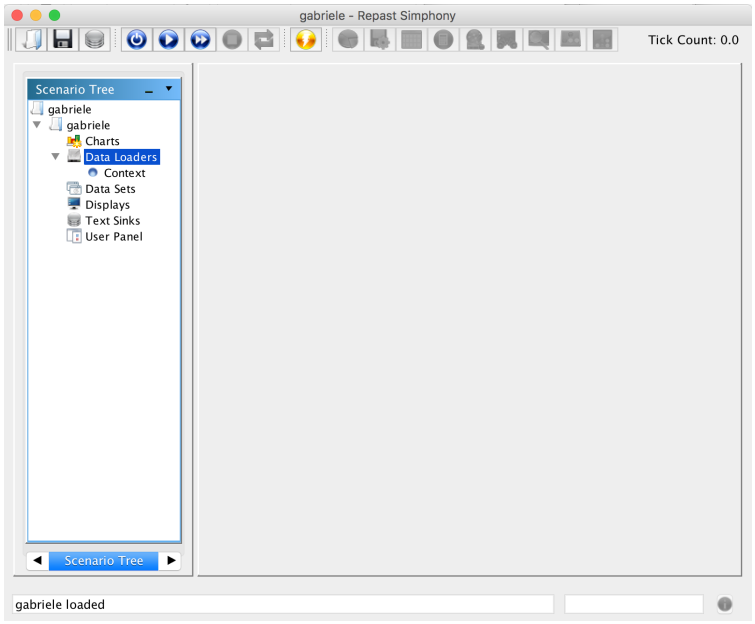


Clicking next will change the window as follows



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 changes permanent.

1.2.2 Running GABRIELE

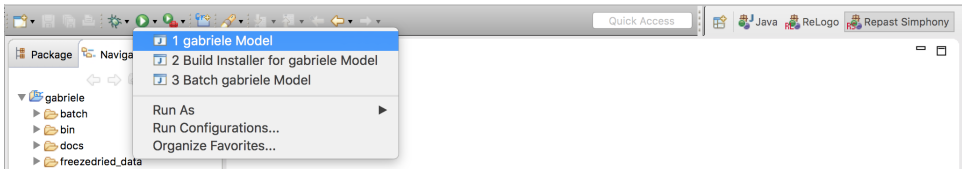
There are two ways of running RS models: The GUI and the BATCH mode.

The GUI mode can be of great visual impact because several monitoring devices continuously updating during runtime 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. 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.



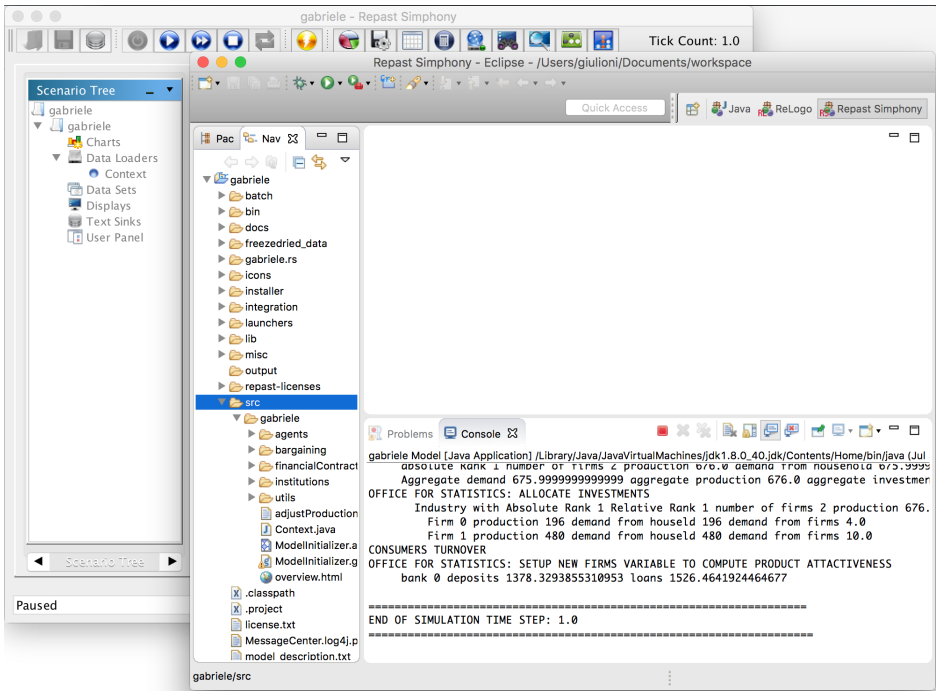
The GUI execution is activated by clicking the `gabriele Model` item, while the BATCH execution can be started by clicking the `Batch gabriele model`.

Running in GUI mode

So, the 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.

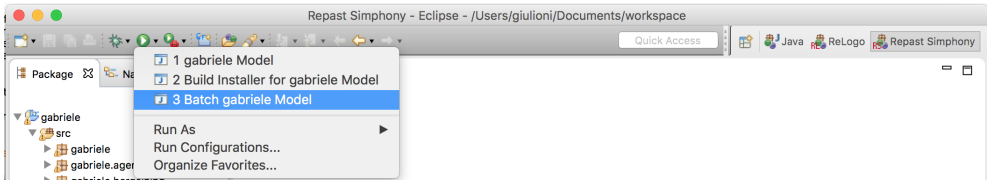


On the other hand, GABRIELE record data in text and csv files. So, running the model they will be added to the model base directory
`/Users/coolcoder/Documents/workspace/gabriele`
Data files can be easily identified because their name starts with `zdata`.

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

Running in BATCH mode

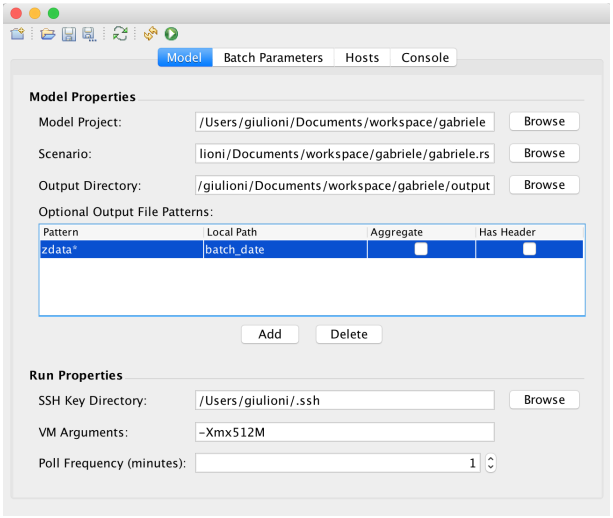
Clicking on the `Batch gabriele model` as in the following figure



activates the batch run configuration wizard.

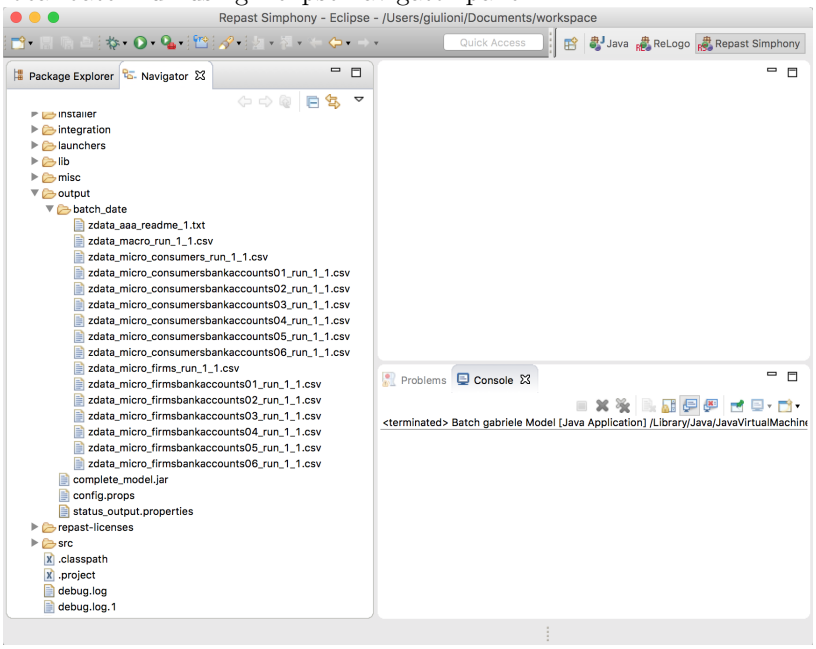
We point the reader to the "Repast Simphony Batch Runs Getting Started" document available in RS website for a full description of the wizard.

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 from all the machines running the model in parallel all the file whose name begins with `zdata` 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.

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



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 repast library and using it directly.

First of all you have to download all the packages which make up the repast symphony library.[1]

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

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

To install repast, make the following steps.

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

```
mkdir repast
```

inside this directory an move into it

```
cd repast
```

Download the files from the Repast update site

```
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>/
         /Users/coolcoder/abm_java_libraries/repast/repast.simphony.runtime_<version>/
  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.

Close the window because we will run the model in BATCH mode only.

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
```

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

Otherwise, 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:

```
/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 wuth 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.

Describing the functioning of a dynamic system implies to know 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 who 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.

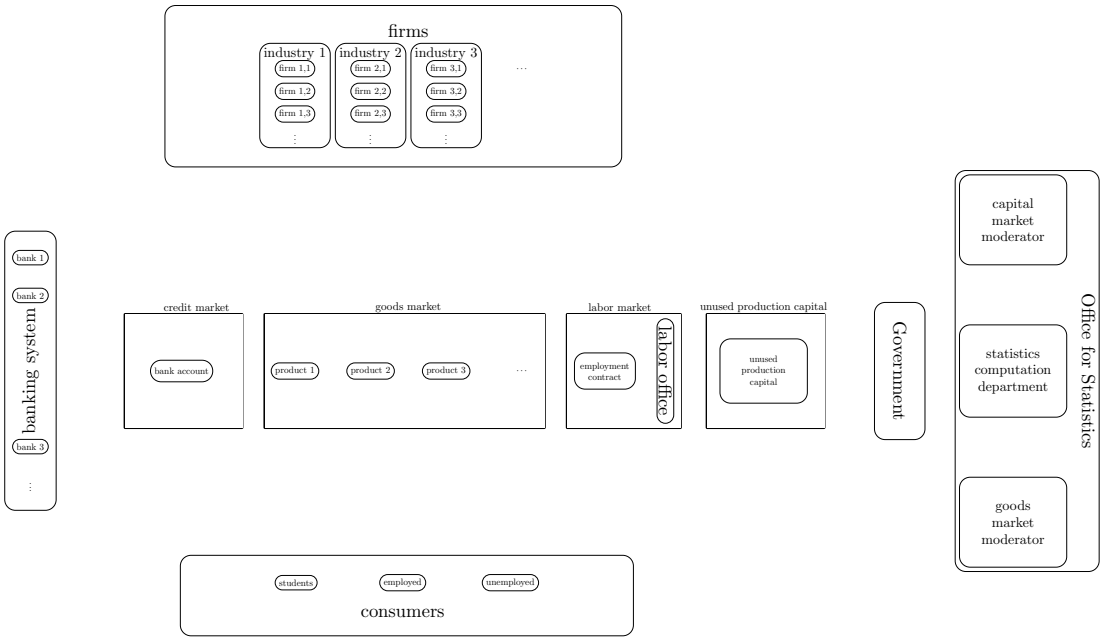


Figure 3.1: Components of the economic system

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.

Multi-instances 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, we obtain the usual categorization of the economy into institutional sectors. Figure 3.2 highlights the institutional sectors in gray.

The other two single-instance agents are highlighted in light blue. The labor office acts in the labor market, while the Office for statistics, have a global view of the system. It computes aggregate variables making them available to all the other agents. Furthermore, it uses this information to moderate the goods and capital markets.

Figure 3.2 also shows the sub-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 sub-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 the same item.

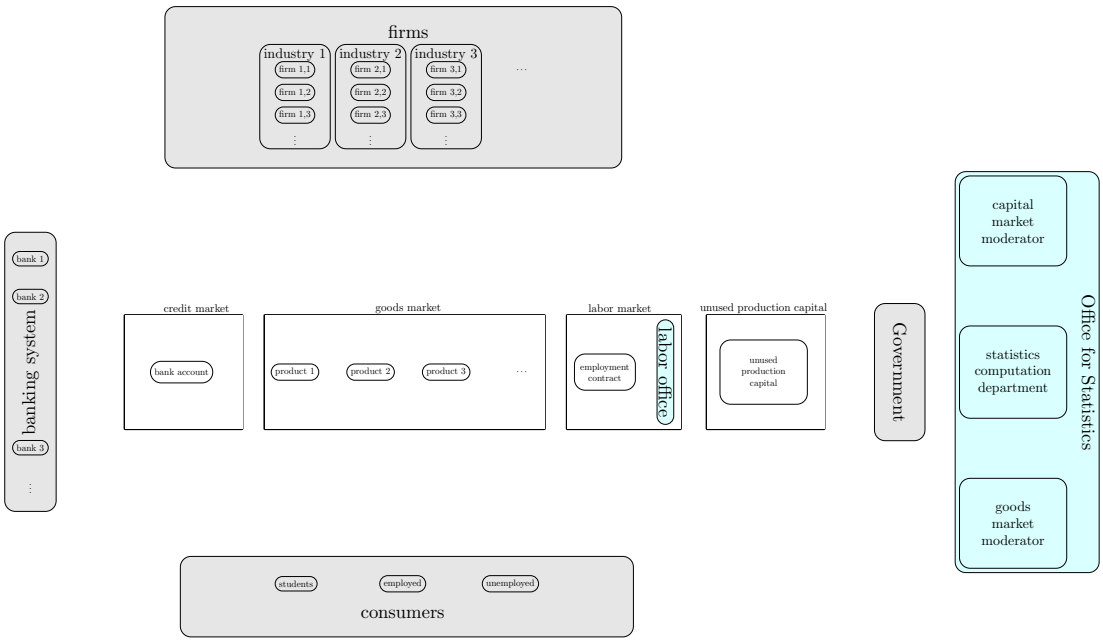


Figure 3.2: Components of the economic system: institutional sectors and other single-instance agents.

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 relative markets. They are highlighted in gray in figure 3.3.

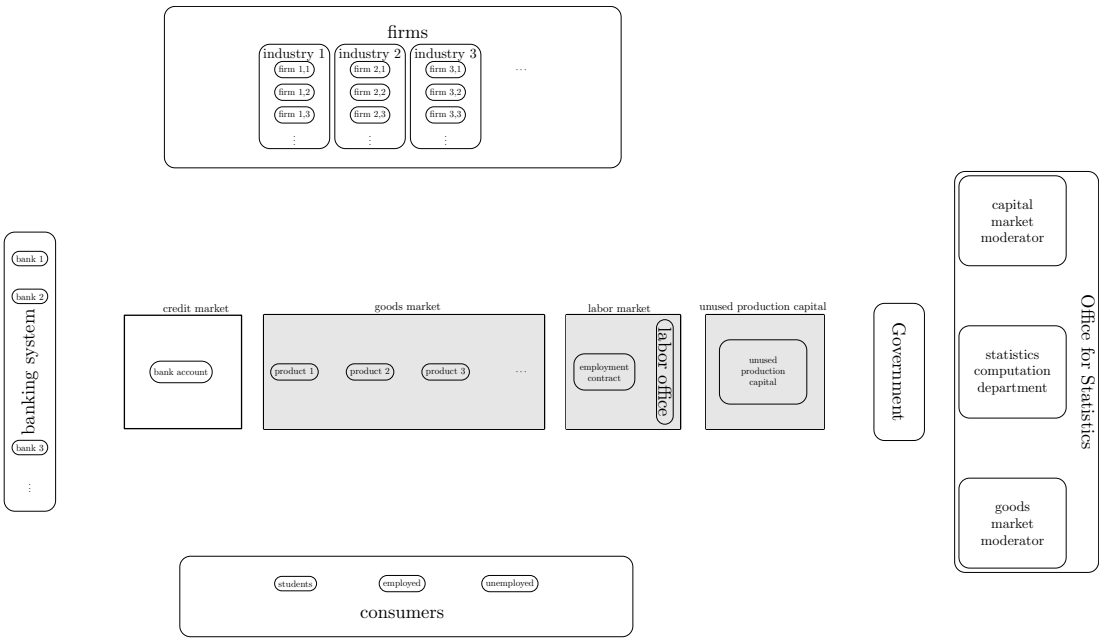


Figure 3.3: Components of the economic system: real markets.

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

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 included in the firms balance sheet. Production capital used in

the production process gradually depreciates. However, it ca 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 ask for new product in the consumption goods market. The latter are then assembled to obtain new production capital. The exceeding production capital that cannot be sold gradually depreciates even thought at a different rate than the production capital employed in the production.

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

3.3 Financial assets

In the previous section we have identified the type of goods present in our model. The identification and specification of financial products to be included in the model has the same importance and is 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.

Every consumer or firm of the model has at leas a bank account. The bank account is a convenient modeling device. It can be used to store wealth, but it is also a mean to obtain credit. In the latter case the amount on the bank account will be negative.

In the present version of the model, we have intermediated finance only. Some agents have positive bank accounts (deposits) while others have negative bank accounts borrowing from the bank.

Financial exchanges are thus performed in the credit market (highlighted in gray in figure 3.4)

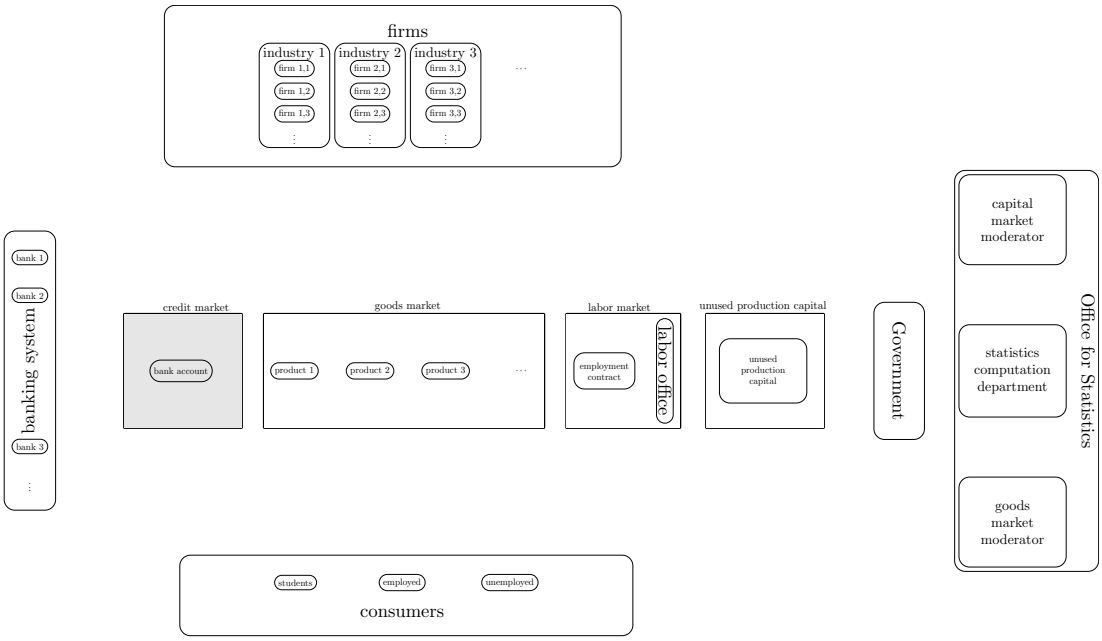


Figure 3.4: Components of the economic system: the credit market.

Considering additional financial contracts, and therefore adding to the model other financial markets, should be done in the near future to let the model gradually approach the reality of nowadays financial markets.

Chapter 4

The Dynamics of Events

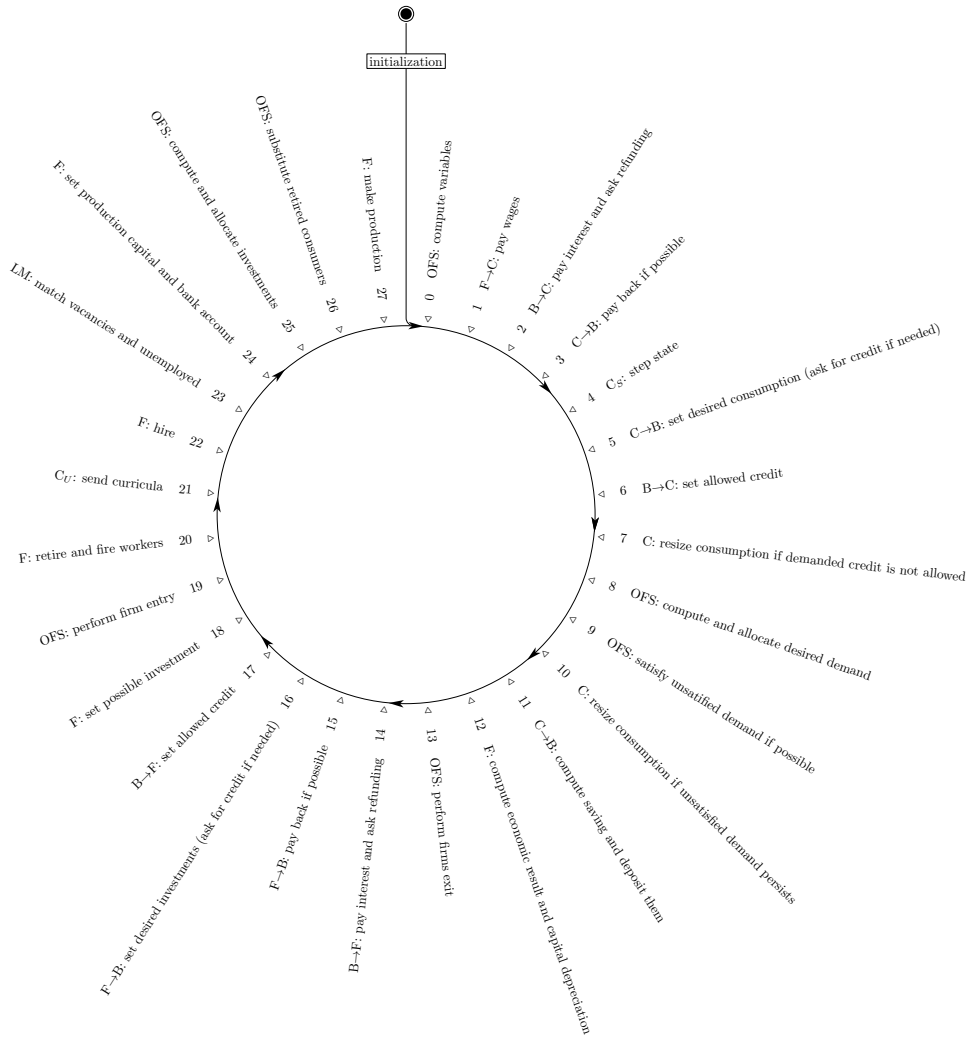


Figure 4.1: sequence of events

4.1 Initialization

The initialization phase allow firms to determine their initial production reducing exogenous setting to the minimum.

When the production is computed, 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 stock flow consistent way. 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:

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

That is, production realized by firm 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:

$$Y_f^{PK} = K_f$$

The potential production of labor is more tricky because workers are heterogeneous and has different productivities. If we identify a worker's productivity with $wp_{w,f}$, we compute the sum of workers productivities

$$wp_f = \sum_w wp_{w,f}$$

the potential production is thus

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

parameter:

in papers	in code	value
θ_{YL}	`parameterOfProductivityInProductionFuncion`	100

To arrive at the very first Y_f we adopt the following strategy.

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

Note how the first step implies a preliminary setup of household working state, while the second step allows the setup of firms assets balance sheet item.

Knowing firms balance sheet assets we set the liabilities in such a way that the balace 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. set agents balance sheet
4. revise agents balance sheet for stock consistency

These phases are explained in details in the documentation ?????. There, we present the steps taken for initializing the model in a sequential order. We also document those method that are not trivial by using UML activity diagrams.

We will now discuss the initialization for each type of agent.

4.1.1 Consumers initialization

Working situation

Students

In an attempt to reduce the number of parameters to the minimum, it is assumed that each worker is characterized by its fundamental ability. It characterize each consumer performance during the education period. The latter in turn determine the productivity as a worker.

We call this parameter the student ability.

parameters:

in papers	in code	value
θ_{as}	`abilityStudent`	`U(Context.minAbilityStudent, Context.maxAbilityStudent)`
θ_{minas}	`Context.minAbilityStudent`	0.35
θ_{maxas}	`Context.maxAbilityStudent`	0.5

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

Next important variable is the consumers age, which is initialized as a random number between zero and a parameter denoting the retirement age.

parameter:

in papers	in code	value
θ_{cea}	`Context.consumerExitAge`	70

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]

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

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

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

probability to be successful.

To initialize the education history we need the following parameters:

in papers	in code	value
θ_{mnfpe}	`maxNumberOfFailedPeriodsOfEducation`	2
θ_{mnpe}	`maxNumberPeriodsOfEducation`	21

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

is reached.

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

The degree of education is assigned as follows:

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

The productivity is assigned as follows. Each year of education increases the consumers ability by a fixed amount. So, the productivity assigned to a non worker with n_{sye} successful year of education is

$$wp = \theta_{as} + \frac{0.5}{\theta_{mnpe}} n_{sye}$$

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

Once the productivity of a non student is known, we compute his/her potential production

$$Y^{Pns} = \theta_{YL} wp$$

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:

in papers	in code	value
θ_{ptbub}	`Context.probabilityToBeUnemployedAtTheBeginning`	0.2

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 a first action to setup the consumers balance sheet.

The involved parameters are:

in papers	in code	value
θ_{nbcc}	`Context.numberOfBanksAConsumerCanBeCustumerOf`	1
θ_{mincba}	`Context.minConsumerInitialBankAccount`	-500
θ_{maxcba}	`Context.maxConsumerInitialBankAccount`	500

Each consumer select randomly a number of banks equal to θ_{nbcc} and open a bank account in each of them. To set the amount of the bank account the code drawn 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 managed when the bank account will be setup). Non negative number are assigner to the bank account amount.

The amount will be revised in the final step of the initialization to ensure 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.

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.

Summing the potential production of each employee, the firm determine the potential production of its labor force:

$$wp_f = \sum_w wp_{w,f}$$

the potential production is thus

$$Y_f^{PL} = \theta_{YL} w p_f$$

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 for each firm is set to

$$K_f = Y_f^{PL}$$

Balance sheet

In the previous step we set up the assets side of the firms balance sheet.

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:

in papers	in code	value
$\theta_{minfieb}$	`Context.minFirmInitialEquityRatio`	0.1
$\theta_{maxfieb}$	`Context.maxFirmInitialEquityRatio`	0.3

Each firm equity is set as follow

$$EF_f = u_{Ef} K_f$$

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

Finally, each firm sets her bank accounts. First of all the level of bank account is computed as

$$BA_f = K_f - EF_f$$

The model has a parameter for the number of banks a firm can be costumer of

parameter:

in papers	in code	value
θ_{nbbc}	`Context.numberOfBanksAFirmCanBeCustumerOf`	1

each firm selects randomly a number of banks equal to θ_{nbbc} and open a bank account in each of them. The amount of the bank account is set to zero (will be assigned in the next step when the bank balance sheet will be initialized), but the variable demanded credit in each bank account is set to BA_f/θ_{nbbc}.

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:

$$L_b = \sum BA_b^-$$

Given each bank assets, we setup the equity by using the following parameters:

in papers	in code	value
$\theta_{minbier}$	`Context.minBankInitialEquityRatio`	0.1
$\theta_{maxbier}$	`Context.maxBankInitialEquityRatio`	0.3

A bank equity base is then computed as

$$EB_b = u_{Eb} L_b$$

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

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

$$D_b = L_b - EB_b$$

Due to the random elements in the initialization of consumers and firms, we usually have

$$\sum BA_b^+ \neq D_b$$

To satisfy the balance sheet identity, the amount of positive bank accounts is adjusted as follows:

$$BA^{+adj} = BA^+ \frac{D_b}{\sum BA_b^+}$$

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

$$\sum BA_b^{+adj} = D_b$$

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

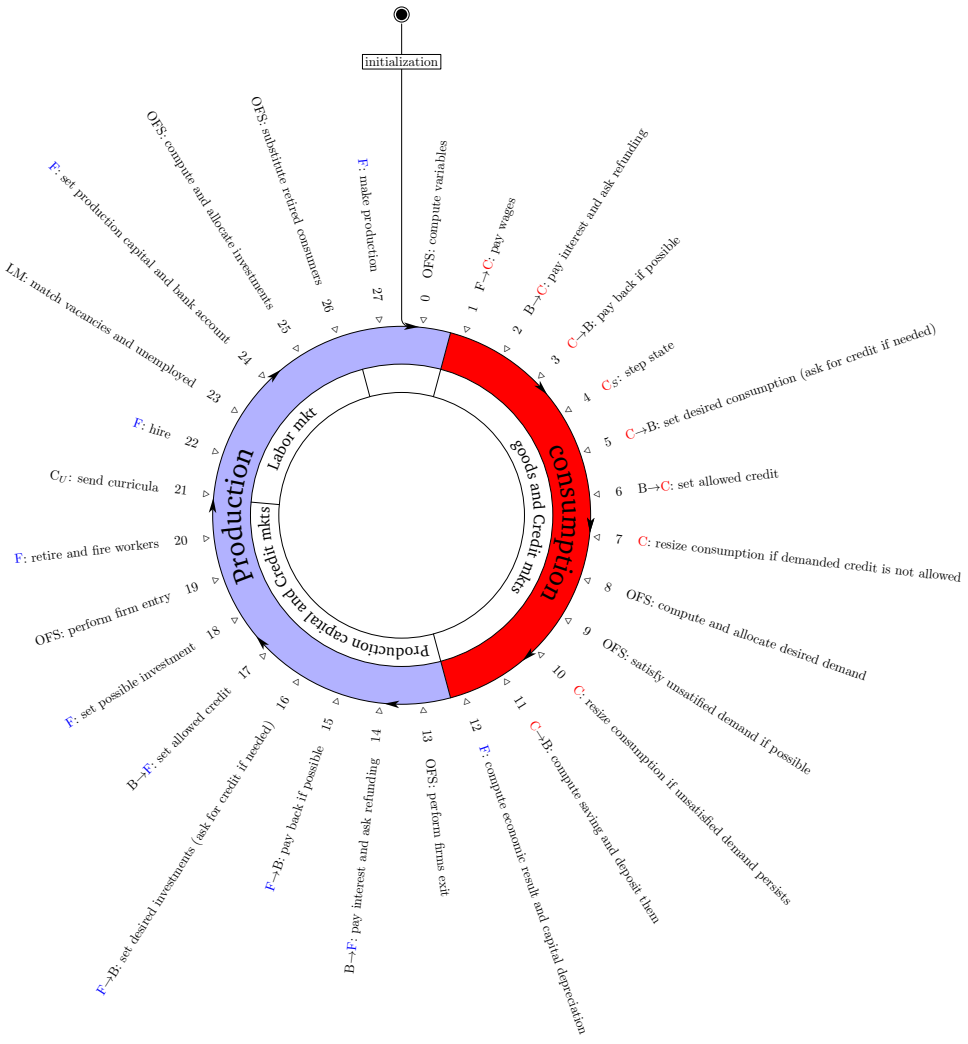


Figure 4.2: sequence of events

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 in the following subsections.

1. consumers receive wages;
2. banks
   (a) account interest
   (b) ask for loan repayments to indebted consumers;
3. consumers refund if possible (if they have enough financial resources);
4. students update their state
5. consumers compute desired consumption and ask for credit in order to achieve the desired level
6. banks decide how much credit to allow;
7. Allowed consumption is computed: consumers adjust their desired consumption according to allowed credit.
8. Goods markets open: Office for statistics helps finding goods.
9. The *effective* level of consumption is established: it may be less or equal to the allowed consumption.
10. consumers adjust their bank accounts (deposits and loans) according to effective consumption.

We now will enter into details of each event.

Consumers receive wages

This step involves only employed consumers. The wage each employed consumer receives is a mark up on the unemployment dole:

$$ww = ud + \theta_{wp} * \theta_{YL} * f(wp) \tag{4.1}$$

where:

in papers	in code	value
ud	`Context.unemploymentDole`	
θ_{wp}	`Context.laborMarketStateToSetWage`	0.5
θ_{YL}	`Context.parameterOfProductivityInProductionFuncion`	100

and $f(wp)$ is a function of workers' productivity.
Three possibilities for $f(wp)$ are presently implemented:

1. $f(wp)$ is the worker's productivity;
2. $f(wp)$ is average productivity of workers having the worker's degree in the employer's firm
3. $f(wp)$ 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:
`Context.wageSettingRule`

The researcher can choose which one to use in a given run by setting the `value` of this parameter in the `batch_parameters.xml` file located in the scenario folder.

As mentioned above, this parameter can take three possible values that are: 0, 1 and 2.

- 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 here

Banks account interests and ask for loan repayments

The accounting of interest updates the households bank accounts.

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

$$BA_{hb,t+1} = BA_{hb,t}(1 + i^+)$$

The model has two different interest rates on loans. The ordinary interest rate (i_w^-) is charged on bank accounts with negative amount owned by workers, while a lower interest rate (i_{nw}^-) is charged on negative bank accounts owned by students and unemployed people.

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 is thus determined as follows:

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

The parameters involved in this computation are:

in papers	in code	value
i^+	`Context.interestRateOnDeposits`	
i^-_w	`Context.interestRateOnLoans`	
i^-_{nw}	`Context.interestRateOnSubsidizedLoans`	
θ_{is}	`Context.percentageOfLoanToRefundForIndebtedWorkersIfAsked`	
pr_{is}	`Context.probabilityToBeAskedToRefundForIndebtedWorkers`	

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. It 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.

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

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

and unsuccessful otherwise.

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

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

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

above:

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

and the subject's productivity is improved as explained in the initialization section:

$$wp = \theta_{as} + \frac{0.5}{\theta_{mnpe}} n_{sye}$$

The subject looses his/her state of student

1. in case of a success, if the maximum periods of possible education ($\theta_{mnpe} + \theta_{mnfpe}$) or
2. in case of failure, if the maximum number of failures admitted (θ_{mnfpe})

is reached. In these cases, his/her status change 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 computes 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^d.

In this model we can have product differentiation. In this case, the desired demand for each item is computed as follows:

$$D_j^d = \alpha_j D^d$$

where α_j is an index of the consumers appreciation for the given item. Of course the condition $\sum_j \alpha_j$ must be verified.

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 the desired amount on one of their bank accounts having null unpaid amount if such account exists. Let us identify this variable with BA^{dc} (where the dc upper script means **d**esired by **c**onsumers). Because they are asking for credit, this amount is negative: $BA^{dc} < 0$. Note that the best bank account con have both a positive and negative amount.

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

- if $BA \geq 0$ and $BA^{ab} < 0$

$$BA^{ab} = \begin{cases} BA^{dc} & \text{with probability} \quad pr_{ab} \\ \theta_{ab} BA^{dc} & \text{with probability} \quad 1 - pr_{ab} \end{cases}$$

- if $BA < 0$ and $BA^{ab} < BA$

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

We have thus the following parameters:

in papers	in code	value
θ_{ab}	`Context.percentageOfCreditAllowedTo ConsumersWhenCreditIsNotTotallyFunded`	
pr_{ab}	`Context.consumersProbabilityToGetFunded`	

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.

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 compute the allowed-desired demand ratio:

$$r_{ad} = \frac{D^a}{D^d}$$

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

$$D^a_j = r_{ad} D^d_j$$

The UML sequence diagram of this event is available here.

Good markets open: office for statistics helps finding goods

We code the goods market function in such a way that both centralized and decentralized matching mechanisms can be mimicked. In particular we make the Office for Statistic to supervise goods allocation.

The task is performed by taking the following steps:

- compute the desired demand
- allocate desired demand to firms
- match demand and supply
- update firms sales

These items are detailed hereafter.

• *Compute the desired demand.* The Office for Statistics computes the demand of each industry by summing over all the consumers. Therefore, provided that in the economy there are J industries, the software computes an array of dimension J: $\{D_1, \cdots, D_J\}$.

The elements of the vector are computed as follows:

$$D_j^d = \sum_c (1 - \theta_{gmi}) D_{cj}^{ac}$$

This formulation is motivated by the attempt to mimic decentralized markets and the deriving imperfections that cause demand get lost in the consumers searching process. We introduce thus the following parameter:

in papers	in code	value
θ_{gmi}	`Context.percentageOfDemandMissed BecauseOfGoodsMarketImperfections`	

The desired aggregate demand is also computed as

$$D^d = \sum_j D_j^d$$

The UML sequence diagram of this event is available here.

• *Allocate desired demand to firms.* In this step, the demand (D_j^d) computed in the previous step is allocated to the various firms according to their share of production in the industry they belong to:

$$D_{fj}^d = D_j^d \frac{Y_{fj}}{Y_j}$$

Note that firms in some industries may have $D_{fj}^d > Y_{fj}$ while in other industries the opposite may hold. These inequalities could be used by firms to guide their production choices.

The UML sequence diagram of this event is available here.

• *Match demand an supply.* In this model we have vertical differentiation. This means that products with higher j are preferred. It may thus happens 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 desire is in short supply.

At the end of this process the demand received by each industry (D_j) and that received by each firm (D_{fj}) is known.

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

$$D_{cj} = D_{cj}^{ac} \frac{D_j}{D_j^d}$$

The UML sequence diagram of this event is available here.

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

$$D_{fj} = D_j \frac{Y_{fj}}{Y_j}$$

This information will be then used to compute profit.

The UML sequence diagrams of this event are available here and here.

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

Once the effective consumption is known, consumers buy the product by and pay them to firms.

The consumers financial position is adjusted accordingly.

Those who consumes less that their income brings saving their saving to the bank with the worst position. Those who were allowed credit decreases 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.

Example

A numerical example of the sequence of action of the above explained process can be reached by clicking here

4.2.2 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 establish the level of production to be sold 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 relate 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 can be moved (depending the chosen parametrization) to other goods by the Office For Statistic. 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.

Currently available inputs can be higher or lower than those needed to realize the target production. Inputs excesses are someway easier to manage than shortages: the production capital used in the production is reduced and workers are fired. The upward adjustment is instead more tricky because several impediments to reach the desired level of inputs can occur. First of all, the level of financial resources including new available credit must be established. Second, the availability of the inputs on the market must be checked. Finally, if shortage of one of the inputs is detected, the demand of the others factors 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. 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. However, production capital, as labor, keeps a part of its production power for the future even if it is not employed in a given period. 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.

Summing up, the process leading to production is composed of the following phases:

- check available financial resources for upward adjustment of production capital. This phases is performed by taking the following steps:
  - compute the economic result and capital depreciation;
  - perform firms turnover;
  - banks account interests and ask for loan repayments to indebted firms;
  - firms refund if possible;

  - the software performs a technical step by resetting some variables of the bank accounts;
  - firms ask for new credit;
  - banks decide how much credit to allow;
  - firms compute investment demand and supply and adjust unpaid amount if possible.
  - entry of new firms that will ask for production capital.

- Labor force adjustment taking in mind the possible financial constraints
  - Firms perform labor force downward adjustment;
  - Unemployeds send curricula;
  - Perform labor force upward adjustment;
- Production capital adjustment taking in mind the possible workers constraint
  - Firms try to adjust their production capital on the market for used production capital;
  - Firms that need additional capital ask for new production goods;
- Consumer turnover
- Make production

We now will enter into details of each step.

Check available financial resources

Compute the economic result and capital depreciation

Each firm economic result is computed as follows

$$CoH_{f,j} = D_{f \leftarrow h,j} + D_{f \leftarrow f,j} - WW_{f,j}$$

where $CoH_{f,j}$ is cash on hand, $D_{f \leftarrow h,j}$ is demand from households, $D_{f \leftarrow f,j}$ is demand from firms and $WW_{f,j}$ is the amount of wages payed by the firm.

Concerning $WW_{f,j}$, it is given by the sum of the individual wage (ww) payed by the firm to the workers it employs. We saw above how ww is computed (see equation 4.1), so we can write

$$WW_{f,j} = \sum ww$$

Cash on hand can be used to integrate production capital if needed. To see if the cash on has to be used to buy production capital, we have to update the sate of this variable taking account of its depreciation.

The production capital of the firm is identified by the K_f variable. We divide this amount into two parts: $D_{f\leftarrow h,j} + D_{f\leftarrow f,j}$ is the production capital that was employed in the production and $K_f - D_{f\leftarrow h,j} - D_{f\leftarrow f,j}$ that was not used. In this model, these two parts of production capital have two distinct depreciation rate ψ_D and $\psi_{!D}$.

The production capital is thus updated as follows:

$$K_f = (D_{f\leftarrow h,j} + D_{f\leftarrow f,j})(1-\psi_D) + (K_f - D_{f\leftarrow h,j} - D_{f\leftarrow f,j})(1-\psi_{!D})$$

The UML sequence diagram of this event is available here.

Perform firms turnover

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

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 $BA_{fb,t} \geq 0$

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

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

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

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:

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

The parameters involved in this computation are:

in papers	in code	value
i^+	`Context.interestRateOnDeposits`	
i^-	`Context.interestRateOnLoans`	
θ_{fbncr}	`Context.percentageOfOutstandingCreditAllowedTo FirmsWhenCreditIsNotCompletelyRenewed`	
pr_{fbren}	`Context.firmsProbabilityToHaveOutstanding DebtCompletelyRenewed`	

The UML sequence diagram of this event is available here

Firms refund if possible

In the previous step banks can ask 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 request 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 recoded.

The UML sequence diagram of this event is available here

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.

If credit is needed, the entrepreneur check 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 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. Because all the bank accounts are evaluated by all firms, the code also identify the worst bank account. It will be used by The residual cash on hand that possibly remains in the following steps will be used to improve the amount of this bank account.

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 of their bank accounts having null unpaid amount if such account exists. Let us identify this variable with BA^{df} (where the df upper script means **d**esired by **f**irms). Because they are asking for credit, this amount is negative: $BA^{df} < 0$. Note that the best bank account con have both a positive and negative amount.

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

- if $BA \geq 0$ and $BA^{abf} < 0$

$$BA^{abf} = \begin{cases} BA^{df} & \text{with probability} \quad pr_{abf} \\ \theta_{abf} BA^{df} & \text{with probability} \quad 1 - pr_{abf} \end{cases}$$

- if $BA < 0$ and $BA^{abf} < BA$

$$BA^{abf} = \begin{cases} BA^{df} & \text{with probability} \quad pr_{abf} \\ BA - \theta_{afb}(BA - BA^{df}) & \text{with probability} \quad 1 - pr_{abf} \end{cases}$$

We have thus the following parameters:

in papers	in code	value
θ_{abf}	`Context.percentageOfNewDemandedCredit AllowedToFirmsWhenCreditIsNotCompletelyAllowed`	
pr_{abf}	`Context.firmsProbabilityToHaveNewDemanded CreditCompletelyAllowed`	

The UML sequence diagram of this event is available here

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

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 exist) 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, we cannot set the amount of investment demand at this stage: firms have first to check for the availability of workers. If workers are not available, the desired level of investment is resized and the demand of investments is set. But this will happen in a later step.

Three variables are set in this step: the `firmInvestment`, the `cashOnHand`, the `promissoryNotes`, the `unpaidAmount`.

Several economic conditions are managed in this step.

In the worst case, the firm suffer a loss and available financial resources are not enough to cover it and a bank that lend money cannot be found. 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 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 appear 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 in that 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>0`, `promissoryNotes=0`, `unpaidAmount=0`

and the case in which the cash on hand is higher than investments:
`firmInvestment < cashOnHand>0`, `promissoryNotes=0`, `unpaidAmount=0`
In the latter case the cash on hand in excess is deposited in the worst bank account.

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 just entered 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:

in papers	in code	value
K_{entry}	`Context.productionOfNewEnteringFirm`	

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.

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 follows 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:

in papers	in code	value
θ_{retire}	`Context.ConsumerExitAge`	

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:

in papers	in code	value
$\theta_{lm-match}$	`Context.FirmsWorkersMatching`	

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:

in papers	in code	value
θ_{retire}	`Context.numberOfJobApplicationAnUnemployedSends`	

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 $\theta_{lm-match}$ curricula, in case 1 they send $\theta_{lm-match} + 1$ 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 $\theta_{lm-match}$ 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

First of all, a further check on the workers production capacity is done. It ca 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.

Through the previous steps we show that a number of firms in the economy want to increase their production capital while other have an excess of it.

In this model, there is a market for exchanging existing unused production capital.

First of all, there is a parameter that transforms unused production capital into reusable production capital. This parameter $\in [0, 1]$.

The sequence of events in the adjustment of 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

The reuse of existing production capital is relevant for credit creation and aggregate demand.

Let discuss the two extreme cases.

If existing capital is not reusable (the parameter is 0), the increase of production capital increase lending or reduce deposits of the firm who made this increase. The balancesheet of the banking sector changes

If existing capital is reusable (the parameter is 1), the balance sheet of the banking sector does not change provided the reusable production capital in the economy is enough.

Consider two firms having both a debt of 100. One of them want to increase its production capital by 10, while the other one has 10 unit of unused capital.

If the second firm can sell the unused capital to the first one, the total amount of lending is unchanged (200) because the debt of the first firm increase to 110 while that of the second firm decrease to 90. In the other case bank lending increase to 210.

However, in the second case the first firm buys new production goods and the aggregate demand is higher than in the first case.

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.

The office for statistics is charged for the provisioning of such goods. Because it also has the role of allocating the aggregate demand from consumers, the easiest way to proceed is that of joining the aggregate demand of consumption good to the aggregate demand of investment goods and then allocate the joined aggregate demand to firms.

The UML sequence diagram of this event is available here and here

Consumer turnover

Consumer turnover is managed in a very simple way. When a consumer reaches the maximum age `Context.ConsumerExitAge` it 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

To realize production firms need two production input: capital and labor.

For each of these two production input we compute the potential production, that is the production that can realize with the current level of the factor provided the other factor is not in short supply.

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

$$Y_f^{PK} = K_f$$

The potential production of labor is more tricky because workers are heterogeneous and has different productivities. If we identify a worker's productivity with $wp_{w,f}$, we compute the sum of workers productivities

$$wp_f = \sum_w wp_{w,f}$$

the potential production is thus

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

finally, the potential output is given by the Leontief type production function

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

Involved parameter:

in papers	in code	value
θ_{YL}	`parameterOfProductivityInProductionFuncion`	100

The UML sequence diagram of this event is available here

Now we are ready to go again through the process that lead 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.

Examples of firm-bank relationship

A numerical example of the sequence of action of the above explained process can be reached by clicking here

A numerical example of the sequence of action of the above explained process can be reached by clicking here

Part III

Modifying GABRIELE

Part IV

Documenting GABRIELE

Chapter 5

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`

5.1 Lists in the table of contents

The `glossaries` package is used to produce the 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.

5.2 Figures

Some figures (for example 3.1 and 3.2) are obtained by using metapost. To modify these figures, you have to edit the corresponding mp source file. Once finished, compile the file with the `mptopdf` command. If the command is not in your system you can either install it or obtain the final pdf by fist compiling with the `mpost` command and the converting the output using `epstopdf`.

Chapter 6

Unified Modeling Language (UML)

Chapter 7

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.