Conducting the Symphony: Behind Repast Wizards

Gianfranco Giulioni
g.giulioni@gmail.com

April 14, 2017

Document in progress ...

Nowadays software are mainly thought to "conduct" users in performing their tasks, however it would be desirable that users can "conduct" the software whenever they want.

Taking advantage of wizards is very useful, but it can block the work if one has no idea on what the wizards do.

These notes aim at explaining what is going on when wizards are used. This knowledge will help in finding a solution when execution does not progress as it should.

1 Installation

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

This can be easily done by following the instruction on the website. However, this implies the installation of eclipse.

The alternative way of using repast proposed in this document avoids the use of eclipse, so that we need to download only the repast java packages.

This can be done from the Repast update site: http://mirror.anl.gov/repastsimphony. In particular by downloading the jar files in the plugins directory.

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

To install repast, make the following steps.

Suppose, for example, you have the home directory `/home/computer`. Create a temporary directory

```
mkdir repast_packages
```

then move to this directory

```
cd repast_packages
```

and 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. Press `ctrl+c` to release the cursor.

Create a new directory where to install the repast. From your home directory `/home/computer/` type
`mkdir repast`
this creates the repast directory, but feel free to name this folder at your convenience.

In the temporary directory `repast_packages`, locate the jar files, and move all these jars in the `repast` directory.

Now, you have to extract all the jars in the `repast` folder. Each jar must be extracted in a directory having the same name of the file (without the jar extension). For example, if you have the `xyz.jar` file, you have to create the `xyz` directory, move the `xyz.jar` into the `xyz` directory and unjar it. Of course, doing this for all the jars is a demanding task.

Fortunately, you can do this at once with the following command

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

Now you can remove all the jar files and the temporary directory.

The result is a series of directory inside the `repast` directory.

Test your installation typing

```
java -cp /home/computer/repast/repast.simphony.runtime_<version>/lib/*:
          /home/computer/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 repast window should pop up.

If you have installed the repast eclipse plugin, you can obtain the same result without performing what described in this section. To start repast from the command line type

```
java -cp <path to eclipse plugins directory>/repast.simphony.runtime_<version>/lib/*:
          <path to eclipse plugins directory>/repast.simphony.runtime_<version>/bin
  repast.simphony.runtime.RepastMain
```

2 Model

2.1 Agent and context: Java

2.1.1 Coding

A minimal model is composed of an agent and a context.

A context is a container of agents. If we want agents to interact, the context must be endowed with structure (projections) that allows agents to do that.

For the moment we just want an agent making a simple task, so that we do not need projections.

Create a new directory for your project, say `myfirst_rs_model`

The directory is now empty.

Using your favorite editor, create the java source code for the agent.

Remember, the file name and the class name must be the same. In the code below, for example you see that the class name is `Random_walker` so, the file name must be `Random_walker.java`.

The file content is as follow:

```
import repast.simphony.random.RandomHelper;
import repast.simphony.engine.schedule.ScheduledMethod;

public class Random_walker {
int position;

public Random_walker(){
position=0;
System.out.println("I am a just created random walker. My position is "+position);
}
@ScheduledMethod(start=1,interval=1)
public void move(){
if(RandomHelper.nextDouble()>0.5){
position++;
}
else{
position--;
}
System.out.println(position);
}
}
```

Now, your directory `myfirst_rs_model` contains the `Random_walker.java` file.

Now we create the java source for the context. Suppose you decide to name it `Random_walker_builder`. Then you have to create the `Random_walker_builder.java` file with the following content

```
import repast.simphony.context.Context;
import repast.simphony.dataLoader.ContextBuilder;

public class Random_walker_builder implements ContextBuilder<Object> {

@Override
public Context<Object> build(Context<Object> context) {
context.add(new Random_walker());
```

```
return context;
}

}
```

Now, your directory `myfirst_rs_model` contains two java files:
`Random_walker.java`
`Random_walker_builder.java`.

2.1.2 Compiling

To describe the compiling process, it is convenient to recall the directories we have used.

In the case presented in this document for example we have repast installed in
`/home/computer/repast`
and the model in
`/home/computer/myfirst_rs_model`

The other relevant information is that, in the java files, we have imported the RandomHelper, ScheduledMethod, Context and ContextBuilder classes. By browsing the repast installation directory, we discover that they are in

```
/home/computer/repast/repast.simphony.core_<version>/bin/
                repast/simphony/random/RandomHelper.class
/home/computer/repast/repast.simphony.core_<version>/bin/
                repast/simphony/engine/schedule/ScheduledMethod.class
/home/computer/repast/repast.simphony.core_<version>/bin/
                repast/simphony/context/Context.class
/home/computer/repast/repast.simphony.dataLoader_<version>/bin/
                repast/simphony/dataLoader/ContextBuilder.class
```

Note how, the final part of paths listed above was included in the import statements.[1] We have thus to inform the compiler about the initial part by including it in the classpath option.

What you have to do is to move the cursor in the model directory and compile the java files with the first part of the paths in the classpath option:

```
javac -cp /home/computer/repast/repast.simphony.dataLoader_<version>/bin:
/home/computer/repast/repast.simphony.core_<version>/bin:. *.java
```

replace `<version>` with your version number (for example `2.3.0`) before pressing the enter key.

In this way you obtain the class files.

[1] You probably know that . in the code represents the / in the file system.

Now, your directory `myfirst_rs_model` contains four files:
`Random_walker.java`
`Random_walker.class`
`Random_walker_builder.java`
`Random_walker_builder.class`.

2.2 Configuration: xml

Five files are needed to configure and then execute your model. Three of them must be named as follows:
`parameters.xml.xml`
`context.xml`
`user_path.xml`
`scenario.xml`

The fifth one can be named by the programmer. More on this later.

The `parameters.xml` file minimal content is as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<parameters>
 <parameter name="randomSeed" displayName="Default Random Seed"
 type="int" defaultValue="__NULL__" />
</parameters>
```

The `context.xml` file gives a name to the Model. Its minimal content is just a line:

```
<context id="myMinimalModel"></context>
```

The `user_path.xml` specify where the class files are located.

If you create this file in the model directory, the content of the file is as follows:

```
<model name="myMinimalModel">
<classpath>
<agents path="./" />
</classpath>
</model>
```

here the `./` means the current directory.

The `scenario.xml` specify the file in which the scenario is described (this is the name of the fourth file)

```
<?xml version="1.0" encoding="UTF-8" ?>
<Scenario>
<repast.simphony.dataLoader.engine.ClassNameDataLoaderAction
      context="myMinimalModel" file="myminimalmodel.xml" />
</Scenario>
```

So in this case the programmer choose to put the scenario description in the `myminimalmodel.xml`. So, this file must be created. There, the programmer put the name of the builder class. One line is sufficient to do that:

```
<string>Random_walker_builder</string>
```

Now, your directory `myfirst_rs_model` contains eight files:
`Random_walker.java`
`Random_walker.class`
`Random_walker_builder.java`
`Random_walker_builder.class`.
`user_path.xml`
`context.xml`
`parameters.xml`
`scenario.xml`
`myminimalmodel.xml`

2.3 Model execution

The class
`repast.simphony.runtime.RepastMain`
is the one which runs models.

To run this class, the classpath must include the runtime lib and bin classes and jars.

The class must be informed on where to find the configuration files. This can be done in two ways:

1. run the class without arguments with the command given in the installation section an then choosing `open` in the `File` menu;
2. providing the configuration folder as argument in the command line:

```
java -cp /home/computer/repast/repast.simphony.runtime_<version>/lib/*:
          /home/computer/repast/repast.simphony.runtime_<version>/bin
  repast.simphony.runtime.RepastMain /home/computer/myfirst_rs_model
```

After a while, the repast window will appear.

Pressing the next button you will see sentences informing about the position of the random walker appear in the console.

The output in the console is similar to

```
I am a just created random walker. My position is 0
1
2
1
```

3 Organizing the files

If you want to have a better structured project you can organize files into folders.

To this aim you can use dedicated tool such as ant or maven. The alternative way proposed in this document is to avoid the use of such tools.

First of all, create a directory for the java source files and move these files into the directory. Usually, the `src` abbreviation is used for source. So, whith the cursor in the project directory type

```
mkdir src
mv Random_walker.java src
mv Random_walker_builder.java src
```

In java, you can give a structure to your classes by using packages. At the beginning of each java file you have to specify to which package the class belongs. For example, if you want the `xyz.class` belong to the `abc.def` package, the first line of the `xyz.java` file is `package abc.def;`

This also imply that the path to the class must be `abc/def/xyz.class` which implies the existence of the directories `abc` and `def`.

The javac compiler can create the structure of directories for you if you include the option `-d dir` where dir is an existing directory. Assume you prompt is in `/home/computer/myfirst_rs_model/src`. Create the `bin` directory:

```
mkdir ../bin
```

Add at the beginning of your two java file the line

```
package randomwalker;
```

Now, compile the java files adding the `-d ../bin` option:

```
javac -d ../bin -cp /home/computer/repast/repast.simphony.dataLoader_<version>/bin:
/home/computer/repast/repast.simphony.core_<version>/bin:. *.java
```

You can verify that the `randomwalker` directory was created inside `bin` and that it contains the two class files.

You can now remove the old class files:

```
cd ..
rm *.class
```

Finally, you can create a directory where to put the xml configuration files. You can give it your favorite name. In this document let call it `scenario`.

with the prompt in the model folder type:

```
mkdir scenario
```

and move all the xml files into the folder:

```
mv *.xml scenario
```

At the end of this process, the structure of `/home/computer/myfirst_rs_model/` is as follows:

```
bin/randomwalker/Random_walker.class
bin/randomwalker/Random_walker_builder.class
scenario/user_path.xml
scenario/context.xml
scenario/parameters.xml
```

scenario/scenario.xml
scenario/myminimalmodel.xml
src/Random_walker.java
src/Random_walker_builder.java

The last important think is to change the configuration (i.e. the xml files) where needed. Remember, the `user_path.xml` file informs the runtime on where the randomwalker package is locates. As specified above, the `user_path.xml` file is in
`scenario/`
and the randomwalker package is in
`bin/`

So, the line
`<agents path="./" />`
of the `user_path.xml` file must be changed in
`<agents path="../bin" />`

Finally, the builder is now in a package, so that the `myminimalmodel.xml` must be changed. Its line:
`<string>Random_walker_builder</string>`
must be changed in
`<string>randomwalker.Random_walker_builder</string>`

Now you have to execute the model. The only change is the location of the xml files that must be specified at the end of the command:

```
java -cp /home/computer/repast/repast.simphony.runtime_<version>/lib/*:
          /home/computer/repast/repast.simphony.runtime_<version>/bin:.
  repast.simphony.runtime.RepastMain /home/computer/myfirst_rs_model/scenatio
```

4 Creating a population

Creating a population can be done with a few lines of code. However we add some more code to have a more informative output in the console.

First of all, we give the agents an identification number and use it in the console output:

```
// +++++++++   begin file Random_walker.java   ++++++++

package randomwalker

import repast.simphony.random.RandomHelper;
import repast.simphony.engine.schedule.ScheduledMethod;

public class Random_walker {
```

```
int position,myIdentificationNumber;

public Random_walker(){
position=0;
System.out.println("I am a just created random walker. My position is "+position);
}

public Random_walker(int id){
myIdentificationNumber=id;
position=0;
System.out.println("I am a just created random walker.
    My identification number is "+myIdentificationNumber+" position is "+position);
}
@ScheduledMethod(start=1,interval=1)
public void move(){
if(RandomHelper.nextDouble()>0.5){
position++;
}
else{
position--;
}
System.out.println("Id "+myIdentificationNumber+" position"+ position);
}
}

// +++++++++   end file Random_walker.java   ++++++++
```

Then we create the population by using a for statement in the `Random_walker_builder.java`

```
// +++++++++   begin file Random_walker_builder.java   ++++++++

package randomwalker;

import repast.simphony.context.Context;
import repast.simphony.dataLoader.ContextBuilder;

public class Random_walker_builder implements ContextBuilder<Object> {

@Override
public Context<Object> build(Context<Object> context) {
for(int i=0;i<3;i++){
context.add(new Random_walker(i));
}
return context;
}
```

```
}
// +++++++++   end file Random_walker_builder.java   ++++++++
```

Of course you can increase the size of your population from 3 to an higher value.

The output on the console should be similar to:

```
I am a just created random walker. My identification number is 0 position is 0
I am a just created random walker. My identification number is 1 position is 0
I am a just created random walker. My identification number is 2 position is 0
Id 2 position -1
Id 1 position 1
Id 0 position 1
Id 1 position 2
Id 0 position 0
Id 2 position -2
Id 1 position 3
Id 0 position -1
Id 2 position -3
Id 1 position 4
Id 0 position 0
Id 2 position -2
```

5 Using Repast essentials

Sometimes we have to obtain information from the model. As an example, consider you want to output the time at which an event occur.

To this aim it is possible to use the `GetTickCount()` method of the `RepastEssentials` class.

Note that the class is in the `repast.simphony.essentials` package. We have to import the class in our code adding the line

```
import repast.simphony.essentials.RepastEssentials;
```

If we want to add the time information when the agent print out the position we have to add the import to the `Random_walker.java` and we have to change the line

```
System.out.println("Id "+myIdentificationNumber+" position"+ position);
```

in

```
System.out.println("Time "+RepastEssentials.GetTickCount()+
       " Id "+myIdentificationNumber+" position "+position);
```

Moreover, the package is not in our building path so that it must be updated as follows:

```
javac -d ../bin -cp
/home/computer/repast/repast.simphony.dataLoader_<version>/bin:
/home/computer/repast/repast.simphony.core_<version>/bin:
/home/computer/repast/repast.simphony.essentials_<version>/bin:. *.java
```

6 Output

To have output from your model you have to set up a data source.

First of all we endow the walker with methods which output information on the position and identification number. It can be done by adding the following code

```
public int getPosition(){
   return position;
}
public int getIdentity(){
   return myIdentificationNumber;
}
```

to the `Random_walker.java` file.

We are now ready to generate a data source.

6.1 Data sources

6.1.1 Aggregate data sources

Prepare a xml file which contains the data source specifications as follows

```
<repast.simphony.data2.engine.DataSetDescriptor>
  <name>average position data set</name>
  <type>AGGREGATE</type>
  <inclTick>true</inclTick>
  <inclBatchRun>false</inclBatchRun>
  <inclRandomSeed>false</inclRandomSeed>
  <scheduleParams>
    <start>1.0</start>
    <interval>1.0</interval>
    <priority>-Infinity</priority>
    <pType>LAST</pType>
    <duration>-1.0</duration>
    <frequency>REPEAT</frequency>
  </scheduleParams>
  <atEnd>false</atEnd>
  <methodDataSources class="linked-hash-map">
    <entry>
      <string>averagePosition</string>
      <repast.simphony.data2.engine.MethodDataSourceDefinition>
```

```
        <id>averagePosition</id>
        <className>randomwalker.Random_walker</className>
        <methodName>getPosition</methodName>
        <aggType>MEAN</aggType>
      </repast.simphony.data2.engine.MethodDataSourceDefinition>
    </entry>
  </methodDataSources>
  <countSources class="linked-hash-map"/>
  <customNADataSources class="linked-hash-map"/>
  <customAggDataSources class="linked-hash-map"/>
</repast.simphony.data2.engine.DataSetDescriptor>
```

Supposing you name the file as `data_source_aggregate.xml`, you have to inform your context to incorporate this new element. This is done by adding the following line to your `context.xml`

```
<repast.simphony.action.data_set
     context="myMinimalModel" file="data_source_aggregate.xml" />
```

So your context file is now

```
<?xml version="1.0" encoding="UTF-8" ?>
<Scenario>
<repast.simphony.dataLoader.engine.ClassNameDataLoaderAction
      context="myMinimalModel" file="myminimalmodel.xml" />
<repast.simphony.action.data_set
      context="myMinimalModel" file="data_source_aggregate.xml" />
</Scenario>
```

6.1.2 Non aggregate data sources

Prepare an xml file as follows:

```
<repast.simphony.data2.engine.DataSetDescriptor>
  <name>id and positions</name>
  <type>NON_AGGREGATE</type>
  <sourceType>randomwalker.Random_walker</sourceType>
  <inclTick>true</inclTick>
  <inclBatchRun>true</inclBatchRun>
  <inclRandomSeed>true</inclRandomSeed>
  <scheduleParams>
    <start>1.0</start>
    <interval>1.0</interval>
    <priority>-Infinity</priority>
    <pType>LAST</pType>
    <duration>-1.0</duration>
    <frequency>REPEAT</frequency>
```

```
  </scheduleParams>
  <atEnd>false</atEnd>
  <methodDataSources class="linked-hash-map">
    <entry>
      <string>Identity</string>
      <repast.simphony.data2.engine.MethodDataSourceDefinition>
        <id>Identity</id>
        <className>randomwalker.Random_walker</className>
        <methodName>getIdentity</methodName>
      </repast.simphony.data2.engine.MethodDataSourceDefinition>
    </entry>
    <entry>
      <string>Position</string>
      <repast.simphony.data2.engine.MethodDataSourceDefinition>
        <id>Position</id>
        <className>randomwalker.Random_walker</className>
        <methodName>getPosition</methodName>
      </repast.simphony.data2.engine.MethodDataSourceDefinition>
    </entry>
  </methodDataSources>
  <countSources class="linked-hash-map"/>
  <customNADataSources class="linked-hash-map"/>
  <customAggDataSources class="linked-hash-map"/>
</repast.simphony.data2.engine.DataSetDescriptor>
```

Supposing you name the file as `data_source_individual.xml`, you have to inform your context to incorporate this new element. This is done by adding the following line to your `context.xml`

```
<repast.simphony.action.data_set
     context="myMinimalModel" file="data_source_individual.xml" />
```

So your context file is now

```
<?xml version="1.0" encoding="UTF-8" ?>
<Scenario>
<repast.simphony.dataLoader.engine.ClassNameDataLoaderAction
      context="myMinimalModel" file="myminimalmodel.xml" />
<repast.simphony.action.data_set
      context="myMinimalModel" file="data_source_aggregate.xml" />
<repast.simphony.action.data_set
      context="myMinimalModel" file="data_source_individual.xml" />
</Scenario>
```

Once the data sources was created, you can use it both for GUI and recording to a file.

6.2 GUI output: Adding a chart

GUI is is not the most efficient way to output results. However, it can be useful for debugging and to impress those who are watching the simulation.

The proposed exercise is to make a chart reporting the average position of the walkers.

Prepare a xml file which contains the chart specifications as follows

```
<repast.simphony.chart2.engine.TimeSeriesChartDescriptor>
  <name>average position chart</name>
  <dataSet>average position data set</dataSet>
  <xAxisLabel>Tick Count</xAxisLabel>
  <yAxisLabel>average position y label</yAxisLabel>
  <chartTitle>average position title</chartTitle>
  <type>TIME_SERIES</type>
  <background>
    <red>192</red>
    <green>192</green>
    <blue>192</blue>
    <alpha>255</alpha>
  </background>
  <gridLineColor>
    <red>255</red>
    <green>255</green>
    <blue>255</blue>
    <alpha>255</alpha>
  </gridLineColor>
  <showGrid>true</showGrid>
  <showLegend>true</showLegend>
  <seriesIds>
    <entry>
      <string>averagePosition</string>
      <SeriesData>
        <label>averagePosition</label>
        <color>
          <red>0</red>
          <green>0</green>
          <blue>255</blue>
          <alpha>255</alpha>
        </color>
      </SeriesData>
    </entry>
  </seriesIds>
  <dataValueIds/>
  <plotRangeLength>-1</plotRangeLength>
</repast.simphony.chart2.engine.TimeSeriesChartDescriptor>
```

Supposing you name the file as `time_series_chart.xml`, you have to inform your context to incorporate this new element. This is done by adding the following line to your `context.xml`

```
<repast.simphony.action.time_series_chart
     context="myMinimalModel" file="time_series_chart.xml" />
```

So your context file is now

```
<?xml version="1.0" encoding="UTF-8" ?>
<Scenario>
<repast.simphony.dataLoader.engine.ClassNameDataLoaderAction
      context="myMinimalModel" file="myminimalmodel.xml" />
<repast.simphony.action.data_set
      context="myMinimalModel" file="data_source_aggregate.xml" />
<repast.simphony.action.data_set
      context="myMinimalModel" file="data_source_individual.xml" />
<repast.simphony.action.time_series_chart
      context="myMinimalModel" file="time_series_chart.xml" />
</Scenario>
```

At the end of this process, the structure of `/home/computer/myfirst_rs_model/` is as follows:

```
bin/randomwalker/Random_walker.class
bin/randomwalker/Random_walker_builder.class
scenario/user_path.xml
scenario/context.xml
scenario/scenario.xml
scenario/myminimalmodel.xml
scenario/data_source_aggregate.xml
scenario/data_source_individual.xml
scenario/time_series_chart.xml
src/Random_walker.java
src/Random_walker_builder.java
```

You can now execute the model and the chart should appear.

6.3 Text output

This is the most convenient way to output the results of the simulation.

Both aggregate data and individual data ca be collected. Data come from the respective data source.

To record aggregate data, prepare a xml file with the following specifications

```
<repast.simphony.data2.engine.FileSinkDescriptor>
  <name>recorded_data</name>
  <dataSet>average position data set</dataSet>
```

```
  <delimiter>,</delimiter>
  <format>TABULAR</format>
  <sourceIds class="linked-hash-set">
    <string>tick</string>
    <string>averagePosition</string>
  </sourceIds>
  <fileName>ModelOutput.txt</fileName>
  <addTimeStamp>true</addTimeStamp>
</repast.simphony.data2.engine.FileSinkDescriptor>
```

Note how the name in the dataSet tag is the one given to the data source: `<dataSet>average position data set</dataSet>`

To record individual data, prepare a xml file with the following specifications

```
<repast.simphony.data2.engine.FileSinkDescriptor>
  <name>individual data</name>
  <dataSet>id and positions</dataSet>
  <delimiter>,</delimiter>
  <format>TABULAR</format>
  <sourceIds class="linked-hash-set">
    <string>run</string>
    <string>random_seed</string>
    <string>tick</string>
    <string>Identity</string>
    <string>Position</string>
  </sourceIds>
  <fileName>ModelOutput_individual.txt</fileName>
  <addTimeStamp>true</addTimeStamp>
</repast.simphony.data2.engine.FileSinkDescriptor>
```

Supposing you name the files as `write_aggregate_data_to_file.xml` and `write_individual_data_to_file.xml`, you have to inform your context to incorporate this new element. This is done by adding the following lines to your `context.xml`

```
<repast.simphony.action.file_sink
     context="myMinimalModel" file="write_aggregate_data_to_file.xml" />
<repast.simphony.action.file_sink
     context="myMinimalModel" file="write_individual_data_to_file.xml" />
```

So your context file is now

```
<?xml version="1.0" encoding="UTF-8" ?>
<Scenario>
<repast.simphony.dataLoader.engine.ClassNameDataLoaderAction
```

```
      context="myMinimalModel" file="myminimalmodel.xml" />
<repast.simphony.action.data_set
      context="myMinimalModel" file="aggregate_data_source.xml" />
<repast.simphony.action.time_series_chart
      context="myMinimalModel" file="time_series_chart.xml" />
<repast.simphony.action.file_sink
     context="myMinimalModel" file="write_aggregate_data_to_file.xml" />
<repast.simphony.action.file_sink
     context="myMinimalModel" file="write_individual_data_to_file.xml" />
</Scenario>
```

At the end of this process, the structure of `/home/computer/myfirst_rs_model/` is as follows:

```
bin/randomwalker/Random_walker.class
bin/randomwalker/Random_walker_builder.class
scenario/user_path.xml
scenario/context.xml
scenario/scenario.xml
scenario/myminimalmodel.xml
scenario/data_source_aggregare.xml
scenario/data_source_individual.xml
scenario/time_series_chart.xml
scenario/write_aggregate_data_to_file.xml
scenario/write_individual_data_to_file.xml
src/Random_walker.java
src/Random_walker_builder.java
```

Note how the text files with the data will be created in the directory you launch the `java` command. The file names are `ModelOutput<date>.txt` and `ModelOutput_individual<date>.txt`

7 Loading Parameters at run time

This is a very important feature.

For example in our case we have the number of random walkers.

Prepare the xml file `parameters.xml` as follows

```
<parameters>
 <parameter name="randomSeed" displayName="Default Random Seed"
       type="int" defaultValue="__NULL__" />
 <parameter name="numberOfWalkers" displayName="Number of walkers"
       type="int" defaultValue="12" />
</parameters>
```

Note that the first parameter tag is compulsory.

Once you have saved this file in the scenario folder you can execute the model. When the runtime appears, you can check the parameters tab to see the parameter you set up.

Now, we are ready to modify the code to read the file.

To this end, edit the `Random_walker_builder.java` and modify it as follows.

```
package randomwalker;

import repast.simphony.context.Context;
import repast.simphony.dataLoader.ContextBuilder;
import repast.simphony.essentials.RepastEssentials;
import repast.simphony.engine.environment.RunEnvironment;
import repast.simphony.parameter.Parameters;
public class Random_walker_builder implements ContextBuilder<Object> {
int numberOfRandomWalker;
@Override
public Context<Object> build(Context<Object> context) {
System.out.println("Time "+RepastEssentials.GetTickCount());
Parameters params = RunEnvironment.getInstance().getParameters();
        numberOfRandomWalker = (Integer)params.getValue("numberOfWalkers");
for(int i=0;i<numberOfRandomWalker;i++){
context.add(new Random_walker(i));
}
return context;
}
}
```

Recompile the java files.

From now on, you can change the number of agents in the `parameters.xml` file without recompiling the code.

8 Batch runs

Batch run gives the possibility to run the model without the GUI. It is a very important features and bulk simulations are performed using this method.

A very important feature is the possibility to run the same model several times. This can be done with given parameters but changing the random seed of the simulation or one can set the random seed and gradually change one or more parameters (parameters sweep).

Three ingredients are needed to run a batch simulation

1. stop the simulation at a given time
2. configure how the batch should behave by writing a batch parameter xml file
3. start the simulation with the appropriate command line

Stop the simulation at a given time. Add the following code

```
if (RunEnvironment.getInstance().isBatch()){
     RunEnvironment.getInstance().endAt(3);
     }
```

before returning the context in your builder class
compile your code.

configuring the batch Prepare an xml file as follows

```
<?xml version="1.0" ?>
<sweep runs="2">
<parameter name="numberOfWalkers" type="constant"
      constant_type="int" value="3"></parameter>
</sweep>
```

```
<parameter name="numberOfWalkers" type="number"
    number_type="int" start="10" end="11" step="1">
```

or

```
<parameter name="numberOfWalkers"
      type="list" value_type="double" values="0.2 0.3">
```

Suppose you save this file in the scenario folder with the name `batch_parameters.xml`.

running the batch To run the batch you have to type

```
java -cp /home/computer/repast/repast.simphony.runtime_<version>/lib/*:
  /home/computer/repast/repast.simphony.runtime_<version>/bin
  repast.simphony.runtime.RepastBatchMain
  -params /home/computer/myfirst_rs_model/scenario/batch_parameters.xml
  /home/computer/myfirst_rs_model/scenario
```

9 Mean field interaction

Suppose you want to model a situation where the walkers have increasing difficulties to move away from the average position of the walkers which are in the system.

So probabilities of moving depend on the distance of the walker's position from the average.

To implement this situation the walkers have to be allowed to access the mean position information. The mean position is computed by the average position data source configured in the xml file. However, how to get this information in the code is unknown. The solution is to compute the average directly in the code and use it.

The steps to do are the following

1. decide how to schedule the computation of the average;
2. write the code to compute the average and schedule the computation;
3. let each random walker to access the average to compute probabilities and use them to determine their position in the next time step.

Step 1 and 2 are related: the code to compute the average depends on the way you will schedule the computation of the average.

There are two possible way to schedule to realize our intent. The first one is to use annotations, while the second one is to manage the schedule directly.

9.1 Scheduling by annotation

We have first to understand how scheduling with annotation works. When scheduling is done by mean of annotations, the action is added to the schedule when the object holding the annotation is added to the context.

This implies that we have to create a class which schedule a method that computes the average. We have to instantiate it in the builder and add the instance to the context. We have to give the class the abilities to compute the average. To do that, we use the methods supplied by the `AggregateDSCreator` Repast class. Among them, there is the `createMeanSource` method that returns an `AggregateDataSource`. We will use an instance of this data source to compute the average.

Summing up we do the following

1. create a class that implements the `NonAggregateDataSource` interface. This class must override the `NonAggregateDataSource` methods.
2. create a class where a method that computes the average is scheduled. In this class the following is done
   - create an instance of `AggregateDSCreator` by using an instance of the `NonAggregateDataSource` in its constructor.
   - obtain an instance of `AggregateDataSource` with a call to `AggregateDSCreator.createMeanSource()`
   - compute the average using the instance of the `AggregateDataSource`
3. instantiate the class where the method which computes the average is scheduled and add it to the context in the builder class.

This is a more structured program and we want to organize it in a structured package.

We name the package `randomwalker` and we organize classes in the following sub-packages:
`randomwalker.agents`
`randomwalker.utils`

We decide to include in the utils package the `NonAggregateDataSource` and the class where the average is computed. We name them as `PositionNonAggregateDataSource` and `PositionAverageAggregateDataSource`.

We then reorganize the `src` as follows:

```
src/randomwalker/RandomWalkerBuilder.java
src/randomwalker/agents/RandomWalker.java
src/randomwalker/utils/PositionNonAggregateDataSource.java
src/randomwalker/utils/PositionAverageAggregateDataSource.java
```

The contents of the files are

```
// +++++++ src/randomwalker/utils/PositionNonAggregateDataSource.java

package randomwalker.utils;

import randomwalker.agents.RandomWalker;

import repast.simphony.data2.NonAggregateDataSource;

public class PositionNonAggregateDataSource implements NonAggregateDataSource{
@Override
public Object get(Object rw){
RandomWalker myrw1=(RandomWalker)rw;
return new Double(myrw1.getPosition());
}
public Class getSourceType(){
return (new Object()).getClass();
}
public Class getDataType(){
return (new Object()).getClass();
}
public String getId(){
return "PositionNonAggregateDataSource";
}

}

//   ++++++++ src/randomwalker/utils/PositionAverageAggregateDataSource.java

package randomwalker.utils;

import randomwalker.utils.PositionNonAggregateDataSource;
import randomwalker.agents.RandomWalker;

import repast.simphony.data2.AggregateDSCreator;
import repast.simphony.data2.AggregateDataSource;
```

```
import repast.simphony.engine.schedule.ScheduledMethod;
import repast.simphony.context.Context;

public class PositionAverageAggregateDataSource{
Double averagePositionOfRandomWalkers;
Context<Object> randomWalkersUtilsContext;
AggregateDataSource averagePositionOfRandomWalkersDataSource;

public  PositionAverageAggregateDataSource(Context<Object> context){
randomWalkersUtilsContext = context;
PositionNonAggregateDataSource myds = new PositionNonAggregateDataSource();

AggregateDSCreator positionDScreator = new AggregateDSCreator(myds);
averagePositionOfRandomWalkersDataSource= positionDScreator.createMeanSource
                                          ("my average position data source");
try{
averagePositionOfRandomWalkers=(Double)averagePositionOfRandomWalkersDataSource.get
  (randomWalkersUtilsContext.getObjects(Class.forName("randomwalker.agents.RandomWalker"
}
catch(ClassNotFoundException e){
System.out.println("Class not found");
}
System.out.println("media "+averagePositionOfRandomWalkers);
}

@ScheduledMethod(start=1,interval=1,priority=1)
public void computeAveragePosition(){
averagePositionOfRandomWalkersDataSource.reset();
try{
averagePositionOfRandomWalkers=(Double)averagePositionOfRandomWalkersDataSource.get(rando
}
catch(ClassNotFoundException e){
System.out.println("Class not found");
}
System.out.println("media "+averagePositionOfRandomWalkers);
}

}

//    ++++++ src/randomwalker/agents/RandomWalker.java

package randomwalker.agents;

import repast.simphony.random.RandomHelper;
import repast.simphony.engine.schedule.ScheduledMethod;
import repast.simphony.essentials.RepastEssentials;
```

```
public class RandomWalker {
int position,myIdentificationNumber;
public RandomWalker(){
position=0;
//System.out.println("Creato alla posizione "+position);
}
public RandomWalker(int id){
myIdentificationNumber=id;
position=0;
//System.out.println("Time "+RepastEssentials.GetTickCount()+" created agent with Id "+my
}
@ScheduledMethod(start=1,interval=1,priority=2)
public void move(){
if(RandomHelper.nextDouble()>0.5){
position++;
}
else{
position--;
}
// System.out.println("Time "+RepastEssentials.GetTickCount()+" Id "+myIdentificationNum
}
public int getPosition(){
return position;
}
public int getIdentity(){
return myIdentificationNumber;
}
}

//    +++++++src/randomwalker/RandomWalkerBuilder.java

package randomwalker;

import randomwalker.agents.RandomWalker;
import randomwalker.utils.PositionAverageAggregateDataSource;

import repast.simphony.context.Context;
//import repast.simphony.context.DefaultContext;
import repast.simphony.dataLoader.ContextBuilder;
import repast.simphony.essentials.RepastEssentials;
import repast.simphony.engine.environment.RunEnvironment;
import repast.simphony.parameter.Parameters;
import repast.simphony.random.RandomHelper;
import repast.simphony.engine.environment.RunState;
import repast.simphony.data2.DataSetRegistry;
import repast.simphony.data2.DataConstants;
```

```
import repast.simphony.data2.DataSetManager;
public class RandomWalkerBuilder implements ContextBuilder<Object> {
int numberOfRandomWalker;
PositionAverageAggregateDataSource myrwu;

@Override
public Context<Object> build(Context<Object> context) {
context.setId("myMinimalModel");
// RandomHelper.setSeed(1234);
RandomHelper.setSeed((int)System.currentTimeMillis());
System.out.println("run "+RunState.getInstance().getRunInfo().getRunNumber()+" randomSeed
//System.out.println(""+RandomHelper.getSeed());
//DataSetRegistry registry = (DataSetRegistry)RunState.getInstance().getFromRegistry(Data
//DataSetManager manager = registry.getDataSetManager(context);
//System.out.println(""+manager);
// System.out.println("Time "+RepastEssentials.GetTickCount());
Parameters params = RunEnvironment.getInstance().getParameters();
        numberOfRandomWalker = (Integer)params.getValue("numberOfWalkers");
for(int i=0;i<numberOfRandomWalker;i++){
context.add(new RandomWalker(i));
}
//System.out.println(""+context.getAgentTypes());

myrwu=new PositionAverageAggregateDataSource(context);
context.add(myrwu);

        if (RunEnvironment.getInstance().isBatch())
          {
            RunEnvironment.getInstance().endAt(3);
           }
return context;
}

}
```

Comparing the two scheduling annotation brings us to an important observation.

The random walkers are scheduled as follows

```
@ScheduledMethod(start=1,interval=1,priority=2)
```

while the computation of the average with

```
@ScheduledMethod(start=1,interval=1,priority=1)
```

The important part is the priority argument. Methods with a higher priority are executed first. So, in our case, the random walkers are updated first

(larger priority: priority=2) and then the average is computed (lover priority: priority=1).

A final note comes from compilation. In this new organization, the java source file are in several folder, so that the command

`javac -d ../bin -cp . . . *.java`

does not compile all the java files.

to have all the files compiled at once we hato to prepare a text file with the paths to the source files:

```
./randomwalker/RandomWalkerBuilder.java
./randomwalker/agents/RandomWalker.java
./randomwalker/utils/PositionNonAggregateDataSource.java
./randomwalker/utils/PositionAverageAggregateDataSource.java
```

Name it as you like, for example `sourcefilespath` and save it in the `src` folder.

Move to prompt to the `src` folder and compiles as follows

`javac -d ../bin -cp . . . @sourcefilespath`.

9.2 Traditional scheduling