The ABMland Toy Model

Nina Schwarz & Daniel Kahlenberg
Department Computational Landscape Ecology
Helmholtz Centre for Environmental Research – UFZ
Permoserstrasse 15
04318 Leipzig
Germany

Technical questions and comments → abmland@ufz.de

Version of this document: August 2011

What is ABMland?
ABMland is a framework for collaboratively developing agent-based models within a spatially explicit and joint environment. It covers three main issues: common scheduling, explicit data exchange, and full functionality even if agents are only partially implemented. ABMland simulates urban land use change and is implemented in Java building upon Repast Simphony. Compared to existing tools, ABMland allows for collaborative implementation of agent-based models and parallel model development while simplifying the coding process.

For information on the overall framework including the installation of the framework, please check the current version of the technical documentation under www.ufz.de/abmland.

What is the ABMland Toy Model?
The ABMland Toy Model 0.0.1 exemplifies the usage of the ABMland framework with a simple example of interactions between Residents and Infrastructure Providers.

Purpose of this document
This document describes the ABMland Toy Model including its installation, the available Java sources and data. Furthermore, it describes the simple implementation example that is provided in the Toy Model and the pre-formatted output.

1 Installing and starting the toy model

> For successfully running the toy model, you need an installation of the ABMland framework. Please follow the technical documentation of the framework for installing the framework before installing the toy model.

The ABMland framework encompasses up to six sub-models: Residents, Infrastructure Providers, Developers, Planners, Business and Lobbyists. Only the implementations for Residents and Infrastructure Providers are available as well as their coupling in the wrapper project.

Thus, three folders are located in the zip-archive of the toy model:

- toywrapper_0.0.1
- toyresident_0.0.1
- toyinfrastructure_0.0.1

1.1 Create plugins and import them as projects

To install the toy model, unpack the zip-archive and move these three folders into your private plugins-folder (see also the technical documentation of the framework). In Eclipse, import each as a project:

1. File – Import: from existing project, root-directory: plugins-folder
2. Select each folder separately
3. Repeat the steps for the remaining projects

1.2 Adapt system paths

1.2.1 toywrapper/toymodels.rs/model.score

Repast S lets you define the physical/file system related conditions – of the contexts and agents defined and used in a project former which inside a file named *.rs/model.score. This basically is an xml file with a hierarchy of nodes, corresponding to how the contexts/agents are organized containing each, certain others in a traditional folder-like hierarchical fashion.

One file is already prepared in toywrapper/toymodels.rs. In the model.score file, the “base path” inside the “implementation” entry (see the xml view of model.score) needs to be changed to the folder with the private plugins (E:\test\myplugins). Do this in level ABMlandModels. On that level the file should look like (at least similar, shown are only the relevant, changeable parts):
Sample “Properties” view of file *.rs/model.score (parts that are subject of change):

```
   IDs
      ...
      Label:: ABMlandModels
      ...
   Implementation
      Base Path:: C:\Users\yourname\.eclipse\yourplugins\plugins
      Bin Directory:: abmlandwrapper_0.0.1\target\classes,common-
repasts_0.0.1\target\classes
      Class Name:: MainContextBuilder
      ...
      Mode:: LOAD # Works for now, AUTO may work too
      Package:: abmland.models.bindings
      ...
```

Please observe that the Label nodes value (“ABMlandModels”) has to pattern-match the value you use for the context entry for that root context in the scenario.xml file. This as well applies to the other agents context values etc., the names chosen once have to be used consistently, it is not enforced prior runtime and runtime errors are harder to translate if you do not know that fact. Furthermore, you should check that the version numbers of ABMland given in the model.score-file match the version you are using.

1.2.2 toywrapper/toymodels.rs/

- **repast.simphony.data.logging.outputter.engine.OutputterInfrastructureAgent .xml**
- **repast.simphony.data.logging.outputter.engine.OutputterResidentAgent.xml**

Absolute paths need to be changed to the private plugins folder to specify the location of the output files to be created. In this case, change the element “fileName” to e. g. “<fileName>E:\test\myplugins\logs\InfrastructureAgentOutputs.txt</fileName>”.

1.2.3 toywrapper/launchers/toymodel.launch

In the Run configurations dialogue, go to “Arguments”. Adapt the working directory to match the following pattern that starts with your private plugins folder:

E:/test/myplugins\abmlandwrapper_1.0.1\target\classes

Press “Apply” to save your changes.

1.3 Run the toy model

1 In eclipse, select the project “toywrapper”.
2 Hit the Run configurations button (green circle with arrow) and select “toymodel”.
3 In the new window, click onto the button on the left with the open folder.
4 In the dialogue, select the folder toymodels.rs as starting point for your simulation run.
5 Initialize and run the models using the GUI buttons.

NOTE: The projects toyinfrastructure and toyresident might show compilation errors, as the projects are not aware of the ABMland framework. If you would like to change the code of the toy models, you have to include the common-repasts_1.0.2.jar provided in the framework installation archive into a lib-folder of your workspace and change the pom.xml of the two projects to match the system paths.

2 Available Java sources and launchers

In short, the following files are provided:

Java sources for **toyinfrastructure** are located in scr/main/java/abmland/models/infrastructure/.

- InfrastructureAgent.java
  Is intermediate between the model and the specific public transport provider agent.
- InfrastructureAgentContext.java
  Needed for the Repast S framework.
- InfrastructureModel.java
  Manages the initialization, data import and export.
- InfrastructureModelContext.java
  Needed for the Repast S framework.
- PublicTransportProvider.java
  Encompasses the actual decision of the agent.

Java sources for **toyresident** are located in scr/main/java/abmland/models/resident/.

- ResidentAgent.java
  Is intermediate between the model and the specific resident type A agent.
- ResidentAgentContext.java
  Needed for the Repast S framework.
- ResidentModel.java
  Manages the initialization, data import and export.
- InfrastructureModelContext.java
  Needed for the Repast S framework.
- ResidentAgentA.java
  Encompasses the actual decision of the agent.

Java sources files for **toywrapper** are located in scr/main/java/abmland/models/bindings/.

- MainContextBuilder.java
  Needed for the Repast S framework.

Other relevant files in the toywrapper project are:

- /launchers/
  - toymodel.launch
    Asks Repast S to start the simulation.
- toymodels.rs/
  - extended_params.xml
    Configuration of the whole simulation. Influences the path to the initialization data of the framework and the models.
  - model.score
    Configuration of the whole simulation. Provides information which models / contexts are part of the simulation.
  - scenario.xml
    Configuration of the whole simulation. Lists all xml files that are used for simulation output.
  - repast.simphony.data.engine.DataSetInfrastructureAgent2.xml
    Defines the data that are used for displaying and storing infrastructure data.
  - repast.simphony.data.engine.DataSetResidentAgent.xml
    Defines the data that are used for displaying and storing resident data.
  - repast.simphony.data.logging.outputter.engine.OutputterInfrastructureAgent.xml
    Defines the output of infrastructure stored to the disk.

- repast.simphony.data.logging.outputter.engine.OutputterResidentAgent.xml
  Defines the output of resident stored to the disk.
- repast.simphony.chart.engine.XYChartInfrastructureAgent2.xml
  Defines the output of infrastructure shown during runtime.
- repast.simphony.chart.engine.XYChartResidentAgent.xml
  Defines the output of resident shown during runtime.

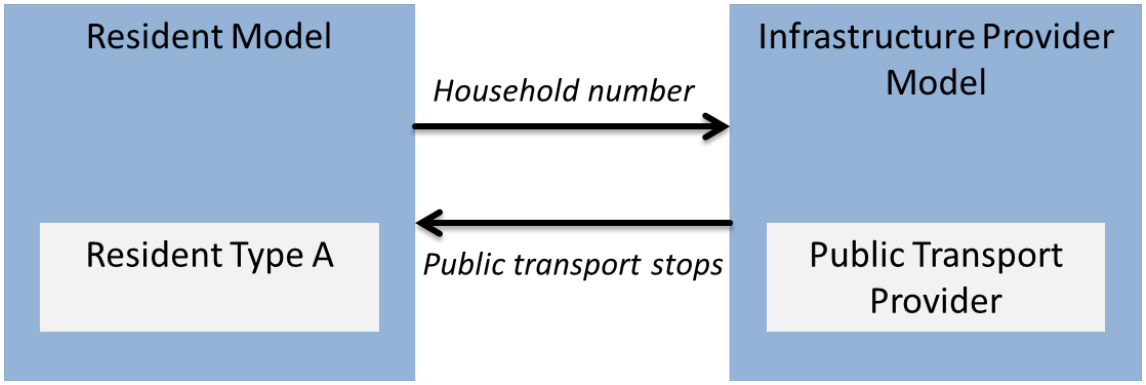
The scenario.xml links to the xml files that are basically created when you first started the Repast S GUI and made additional entries in the simulation tree view there, as for instance data set definitions. Each of those entries will have an xml file reflecting its state. What we did was to prepare some entries that are read initially when you start the simulation run, directed by the entries in the scenario.xml file.

Note: Be careful if you copy runtime (Repast S) generated xml data to this location as it uses free names. Better practice is to learn the syntax and to prepare these data files in xml.

3 Simple interaction of Residents and Infrastructure Providers

Two models interact: Residents and Infrastructure Providers.

- Residents consist of one agent type: type A. This agent represents all households of that type, and households are located on grid cells.
- Infrastructure consists of one agent responsible for the whole area and provides public transport stops (yes/no per cell).



Residents start with an initial distribution of households per grid cell, and the infrastructure provider start with an initial distribution of public transport stops.

In each time step,

- Residents randomly change their number per cell and
- Infrastructure provider builds or reconstructs public transport stops – depending on the imported household number: If the total number of households divided by the current number of public transport stops
  - exceeds 600, then a new public transport stop is built,
  - is smaller than 500, then an existing public transport stop is dismantled.

4 Description of initialization data

Spatially explicit data have 30x30 cells. Using the ETRS LAEA projection, the toy landscape lies in the city of Leipzig, Germany. Cells have a size of 100 x 100 m².

The overall **framework** is initialized using data located in common-repasts/src/main/resources/config/maps/GRID30BY30/2010/.

- Land use
    - Data file: toylanduse30_30.asc
    - Three CORINE land cover classes are present:
        - 111 Continuous urban fabric
        - 112 Discontinuous urban fabric
        - 241 Annual crops associated with permanent crops
- Land price
    - Data file: toylandprices30_30.asc
    - 0 and 1 are randomly distributed.

The name of the initalisation files are given in src/main/resources/config/params/GRID30BY30/2010/MiscConf.xml, LanduseConf.xml and MarketConf.xml.

The **infrastructure** model is initialized using data located in src/main/resources/config/maps/GRID30BY30/2010/

- Initial distribution of public transport stops
    - Data file: toyinfrastructure_30_30.asc
        - -9999: no public transport stop on cell
        - 1: public transport stop on cell
    - 5 public transport stops are initialized

The name of the initalisation file is given in src/main/resources/config/params/InfrastructureConf.xml

The **resident** model is initialized using data located in src/main/resources/config/maps/GRID30BY30/2010/

- Initial distribution of households
    - Data file: toyresident_30_30.asc
        - The number of households per cell is randomly distributed.
        - Initially, households do not dwell on agricultural cells.
    - 3059 households are initialized

The name of the initalisation file is given in src/main/resources/config/params/ResidentConf.xml

5 Model output

The toy model provides three types of output:

1. **Log messages** are stored in [private plugin folder]\toywrapper_0.0.1\target\classes\logs

2. Two **comma-separated files** for results of resident and infrastructure models are stored on disk (location is provided in repast.simphony.data.logging.outputter.engine.OutputterInfrastructureAgent.xml and repast.simphony.data.logging.outputter.engine.OutputterResidentAgent.xml).

3. **Graphs** during runtime are provided for total number of households (configured in repast.simphony.chart.engine.XYChartResidentAgent.xml) and total number of public transport stops (configured in repast.simphony.chart.engine.XYChartInfrastructureAgent2.xml).