# ANGERONA - Manual

Tim Janus, Patrick Krümpelmann
Technische Universität Dortmund, Germany

November 5, 2014

**Abstract**

This Manual explains how the ANGERONA framework can be used by a user. Section 1.1 explains how the binary distribution of the ANGERONA framework is installed and how the ANGERONA framework can be used to perform experiments, therefore a simulation is loaded and explained step by step. In Section 1.2 the external tools for running an ASP solver are installed, such that every scenario can be investigated that is in the binary bundle of ANGERONA. The last Section 1.3 explains how a developer can extend the ANGERONA framework, it explains how the development prerequisites of the ANGERONA framework are installed and which steps are needed to get the ANGERONA framework running from within an IDE. It also shows important points in the framework that allow the developer to create and use a plug-in.

# Contents

# Chapter 1

# Installing ANGERONA

This Chapter explains how ANGERONA can be installed, therefore it starts by using the ANGERONA framework from the perspective of an *user*. That means the latest binary distribution of the framework is fetched and used to perform experiments and investigate configuration files. Then this Chapter explains how the ANGERONA framework can be installed using git as source control system, such that the reader gets prepared to use the framework from the perspective of a *plug-in developer* or a *core developer*.

## 1.1 Using ANGERONA

This Section explains how ANGERONA can be used to perform experiments in multi-agent environments. It assumes that the binary archives are used to install the ANGERONA framework. In Section 1.1.1 the installation of the binary packages of the ANGERONA framework is explained and then Section 1.1.2 shows how those packages can be used to perform a simulation.

### 1.1.1 Install

To install the ANGERONA framework get the latest binary packages from:

-> `http://marathon.cs.tu-dortmund.de/angerona/latest`

then extract the archive, or alternatively if you want to become a developer use git to clone the Angerona repository and integrate it in your favorite IDE, as described in Section 1.3.

The content of the archive runs out of the box, so either execute *Angerona.bat* (Windows) or *Angerona.sh* (Unix / MAC) in the extracted folder to start the ANGERONA framework.

If the ANGERONA framework does not start without errors try the following steps:

1. Read the FAQ:
   `https://marathon.cs.tu-dortmund.de/trac/`

2. Post a ticket with error description and system properties on:
   `https://marathon.cs.tu-dortmund.de/trac/newticket`

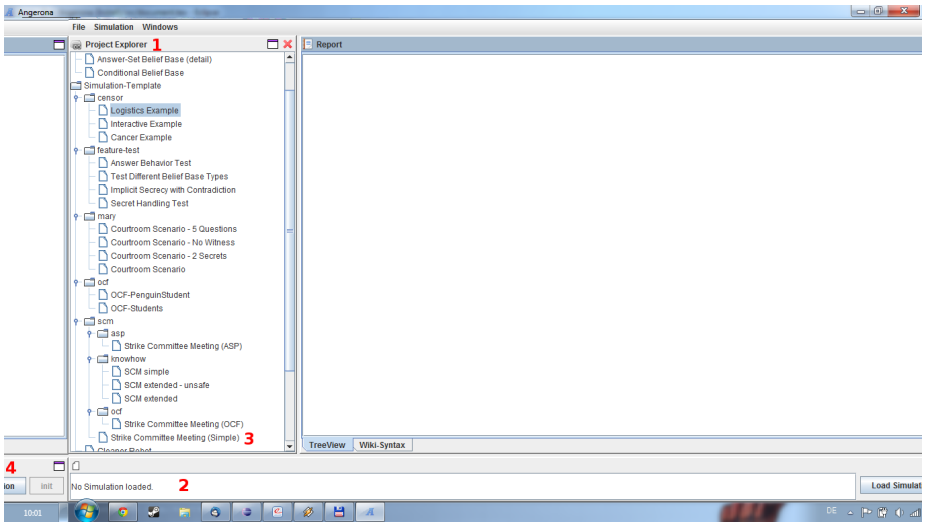


Figure 1.1: ANGERONA GUI - After Start

Figure 1.1 shows the GUI of the ANGERONA framework after the start. The widget labeled with (1) is the *Project Explorer*. It shows files that are listed in the *config* and *examples* folders. The control labeled with (2) outputs the status of the simulation, until now no simulation is loaded. After a double click on the tree node that represents a simulation and is labeled with (3) the text in (2) changes and indicates that the selcted simulation is ready. The second button at the position labeled with (4) becomes useable and initalizes the selected simulation if it got clicked.

### 1.1.2 Perform a Simulation

After the ANGERONA framework is started and the simulation *Strike Committee Meeting (Simple)* is loaded a click on the button labeled with *init* creates the environment for the simulation and changes the view, such that it looks like in Figure 1.2.

The *Project Explorer* becomes a background tab and on position (1) on Figure 1.2 the *Entity Explorer* is shown. The *Entity Explorer* contains every agent in the environment and shows the agents' components, such as its belief base that represents the agent's world knowledge or belief bases that represent the agent's view on other agents. It also shows every data component used by the agent like the *Secrecy Knowledge* or the *Action History*. We take a closer look on the functionality of the *Entity Explorer* later. Position (2) on Figure 1.2 shows the *Report View*, which is used to inform the user of the ANGERONA framework about the steps the agents have performed in the simulation so

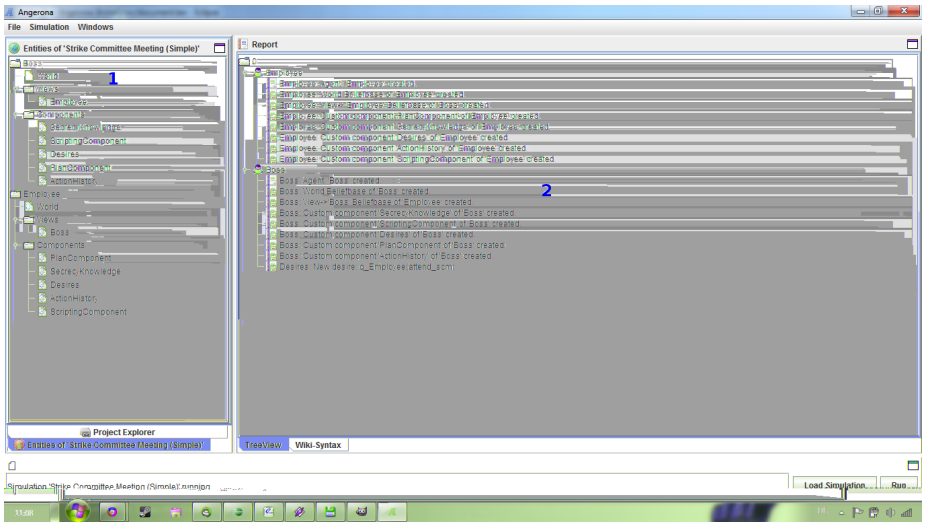


Figure 1.2: Simulation Initalized

far. Currently there are only messages about several data components that are created in simulation tick zero. The simulation tick zero refers to the simulation intialization phase. To get a more informative output in the GUI click on the run button until it is labeled with finish, such that the GUI looks like in Figure 1.3.

Figure 1.3 shows the entire simulation tick one. Position (1) and (2) summarize the agent's cycle in the tick by showing the action that is performed by the agent. The boss queries the employee if he attends the strike committee meeting and the employee answers with no (false). The report entry as position (3) indicates that the employee has thought about answering with yes (true). The violates operator used to perform an action mentally returned that a secret would be revealed by answering with yes and therefore the employee decided to lie. The components in the *Entity Explorer* can be double clicked to open a more detailed view on them, such that a double click on the employee's view on the boss's beliefs (4) opens a view, that looks like in Figure 1.4.

Now the *Report View* becomes a background tab and a tab that showes the employee's view on the boss's beliefs becomes the foreground tab. This tab has two parts, position (1) on Figure 1.4 shows the navigation control panel, which allows to navigate through different versions of the belief base. Currently the control shows entry 2/5 and the button on position (2) can be used to navigate to the next entry. In entry 2/5 the boss beliefs got a new literal *attend_scm* on position (3). The literal is colored green because it is new in this version of the belief base. If a literal is colored black it is the same as in the previous version, but if a literal is colored red it is no element of the belief base anymore but it is an element of the previous belief base. The same literal is also shown on position (4), this is no redundant information but an information with another

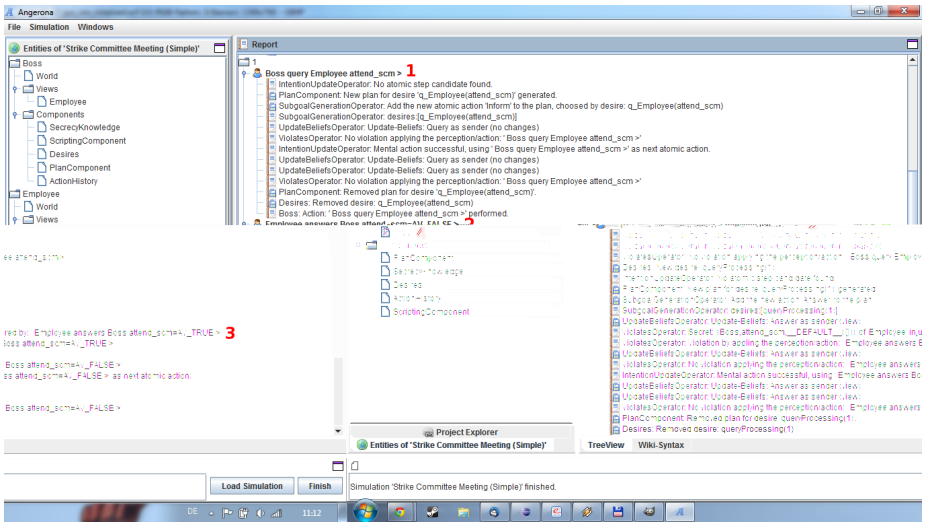


Figure 1.3: Simulation Finished

semantic as the literal on position (3). The literals show on position (4) are the deducted knowledge of the belief base that is shown in position (3). Position (5) shows the operator callstack, which shows the operators that are called until the belief base got changed. The Figure shows that intention update is called from the agent cycle. Intention update called violates, violates called itself and then finally the update beliefs operator is called to mentally apply the action. The belief base shown in Figure 1.4 is the belief base that caused the report output on position (3) in Figure 1.3.

For further details take a look on the XML files in the *examples* and *config* folders.

If you try to load other scenarios like the *Strike Committee Meeting (ASP)* scenario you will receive an error message because those scenarios depend on ASP solvers that are not bundled with the ANGERONA binary package. Section 1.2 explains how the ASP solvers can be installed.

## 1.2 Installing Optional ASP Solvers

As mentioned in the above Section some of the scenarios use disjunctive logic programs under the answer set semantic (ASP) as knowledge representation mechanism and therefore have to invoke external programs to process the answer sets. Because of license issues the ASP solvers cannot be bundled in the binary distribution of ANGERONA. But the solvers are free for academic and educational use and for non-profit organizations. The following list shows the solvers that are compatible with the ANGERONA framework.

-> `http://www.dlvsystem.com/dlv/` - dlv

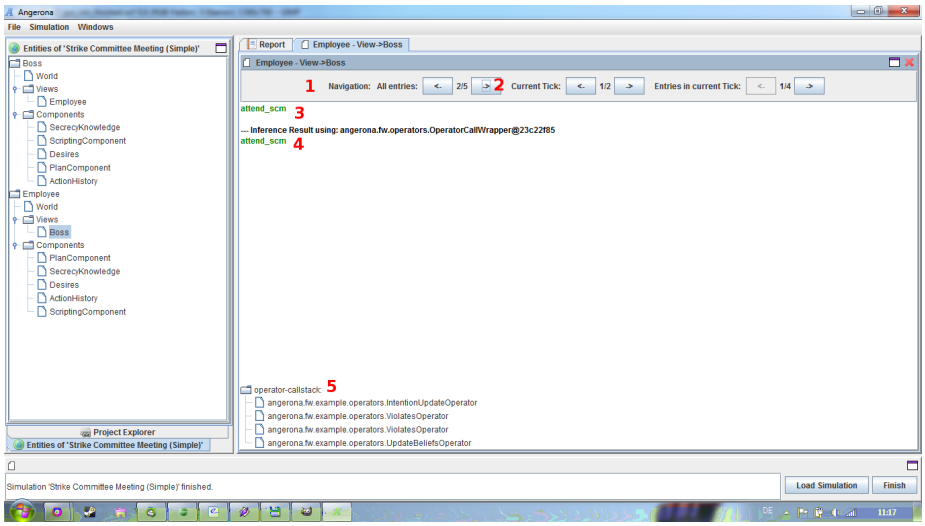


Figure 1.4: View on the Boss's Beliefs

-> `https://www.mat.unical.it/dlv-complex` - dlv-complex
-> `http://potassco.sourceforge.net/` - Clingo

The scenarios mostly use the dlv version of the ASP solver. The downloaded binaries can be saved somewhere on the file system. After the files are downloaded and extracted or installed *config/configuration.xml* needs several adaptations, such that the paths to the ASP solvers are correct.

Imagine User $A$ has a file *dlv-v1.0.exe* on his windows machine in the folder *C:/ASP-Solvers/*. This file is an executable of the DLV solver. The following entry in the configuration.xml gives ANGERONA the ability to find the file:

```
<parameter name="path-dlv" value="C:/ASP-Solvers/dlv-v1.0" />
<!-- or alternatively !-->
<parameter name="path-dlv" value="C:/ASP-Solvers/dlv-v1.0.exe" />
```

If a executable file is not found ANGERONA tries to find the file by appending the system-specific file ending for executable files to the filename and therefore the upper *parameter* tag also works. On Unix systems the file need the $X$ - *execute* permission. Only change the *name* attribute if you really know what you do, otherwise ANGERONA cannot map the parameter-name and therefore cannot decide what the parameter value means.

After the adaption of the *config/configuration.xml* the scenario *Strike Committee Meeting (ASP)* shall run without any errors and has the same structure like *Stirke Committee Meeeting (Simple)*.

## 1.3 Prepare a Working Machine for Development in the ANGERONA Framework

This Section describes how you can contribute to the ANGERONA project. ANGERONA can be developed in several Java IDEs and is tested in Net Beans 7 and several Eclipse versions (Indigo, Juno, Kepler). This Section contains several remarks for windows user, who additionally have to adapt path settings etc. Nevertheless this is a short explanation to get ANGERONA running in a IDE and the Section only briefly explains concepts, such that it is highly recommended to also read the technical report [1] at `http://marathon.cs.tu-dortmund.de/trac/wiki/technical-report`. Section 1.3.1 explains what prerequisites have to be installed and what are the tasks of those prerequisites. Then Section 1.3.2 introduces the Tweety libraries, an important dependency of the ANGERONA framework. It is proposed that every ANGERONA developer also uses the SVN source version of Tweety. Then Section 1.3.3 explains how Maven has to be configured and and shows how some errors occurring through the incomplete Maven integration into eclipse can be fixed or ignored, such that ANGERONA is runnable from the IDE. The last Section 2.1 explains how a developer can extend the ANGERONA framework by providing plug-ins.

### 1.3.1 Install Prerequisites for Development

First of all we have to install and configure a few software packages. We need to install the following list of software:

1. **Java JDK $\geq$1.7** - Should already be installed on most systems. If this is not the case get it from Here.
2. **git client** - In an unix / mac environment install the commandline git client using your package manager. When using Windows we propose to use Tortoise Git as git client.
3. **Maven - External** - In an unix / mac environment install the apache-maven package. When using windows you have to download apache-maven manually from the Homepage. Extract the zip archive somewhere on the harddrive.

   **Windows specific**: To use maven in the command line we have to add the maven binary directory to the Windows Path environment variable. Click onto *Start* $\rightarrow$ *System settings* then select System in the system settings dialog and a new dialog appears containing an advanced System Settings tab. This tab contains a button with the text 'Environment variables'. After clicking on this button a new dialog opens which contains controls for adding and changing environment variables.
   Append the path to the "mvn.exe" to the System environment variable 'Path'.

4. **Java IDE** - Install the Eclipse or Net Beans IDE if not already done. If Eclipse is choosen as IDE then ANGERONA needs the INDIGIO release version or a newer version. Now check if the Eclipse installation is using a JDK. Select *Windows* $\rightarrow$ *Preferences* in Navigation tree of the new dialog select *Java* $\rightarrow$ *Installed JREs* if the used JRE is not a JDK JRE add a JDK JRE to the list and use it as default, version 1.7 is needed. Net Beans is only tested with a version $\geq 7$ but as the Maven integration in Net Beans is better the installation of the prerequisites is finished for Net Beans developer at this point.

   **Windows specific**: Ensure that the JAVA_HOME environment variable points to the installation directory of the JDK selected by eclipse.

5. **Maven - Eclipse Plugin** - Angerona uses Maven for building its subprojects. It is highly recommended to install the Maven Plugin into Eclipse, such that Maven and Eclipse can interoperate without any problems: Select *Help* $\rightarrow$ *Install new Software...* in the Menu.
   The Indigo-Release Site http://download.eclipse.org/releases/indigo contains the Maven Plugin package. Installing the following packages is recommended:
   *Collaboration* $\rightarrow$ *m2e - Maven Integration for Eclipse*
   *Collaboration* $\rightarrow$ *m2e - slf4j over logback logging (Optional)*.

6. **JavaCC - Eclipse Plugin** - You have to install the JavaCC Plugin for Eclipse. Click onto the menu: *Help* $\rightarrow$ *Install new Software...*. Add a new update site with the URL: `http://eclipse-javacc.sourceforge.net/`. Select the update site and then select **SF JavaCC Eclipse Plug-in feature** and install it.

7. **JavaCC m2e Connector** - You have to install a connector which informs

### 1.3.2 Optional: Work with the Tweety Source Code

The Tweety library is a collection of various Java libraries that implement approaches to different areas of artificial intelligence. In particular, it provides different knowledge representation formalisms such as classical logics, conditional logics, probabilistic logics and logic programs under the answer set semantics. Both knowledge representation mechanism, ASP and OCFs, used in the ANGERONA framework use Tweety as dependency, so it highly recommended that one who plans to get into the ANGERONA framework also gets an insight into the Tweety library.

The Tweety library is hosted at Sourceforge:
`http://sourceforge.net/projects/tweety/` it can be checked out using a SVN client. After it has been checked out run the third-party install script in the *thirdparty* folder and import the pom.xml files as projects into your IDE (Eclipse Developers have to use *Import as Maven Project*). Tweety is a collection of libraries and therefore does not contain specific application projects that can be run to check if Tweety works, therefore the project *tweety-maven-build-all* can be run with maven using the goals 'clean install' and shall run through without an error. After this step the local maven repository contains the tweety artifacts.

**Remark**: The first complete build will take some minutes. Maven will download all the plug-ins which are used by the Tweety libraries.
If Maven complains that it cannot find the Java Compiler ensure you have select a JDK JRE as default JRE in Eclipse.

### 1.3.3 Fetch the ANGERONA Repository and Prepare Maven

After installing the software we need to checkout the public git repository of ANGERONA by using the following commands:

```
mk <git-repos-dir>
cd <git-repos-dir>
git clone https://github.com/Angerona/angerona-framework.git

<git-repos-dir> = directory for git repositories
<un> = username
```

As the URL indicates Angerona is hosted by github and if you want to contribute to Angerona you also need a github account. You can either ask us for push privileges or use a fork and send us a pull request for your changes.

**Windows specific**: Create a folder and right click onto the folder: Select *git clone...* in the context menu. Fill the dialog text-fields with the url and directory above.

Now add the ASP-Solvers into the folder
*<angerona-dir>/software/app/src/main/tools/asp-solvers*
therefore refer to the readme.txt in the directory.

Import all the projects in *<angerona-dir>/software*.

**Remark**: Every text file in the ANGERONA framework is encoded in UTF-8. This is the default on most linux systems. Nevertheless you should check your IDE is using UTF-8 encoding in text files. For Eclipse clicking on *Window* $\rightarrow$ *Preferences* and selecting *General* $\rightarrow$ *Workspace*. Choose the UTF-8 text file encoding if it is not already choosen.

Now we have to set the global maven settings. Open the folder *angerona-dir/maven*. There is a text file called settings.xml which contains a configuration

goal type “clean install”. Now click onto run to perform a clean build process and after that the install build process which will install all maven artifacts of ANGERONA into the local Maven repository.

**Remark**: The first complete build will take some minutes. Maven will download all the plug-ins which are used by the ANGERONA framework.
If Maven complains that it cannot find the Java Compiler ensure you have select a JDK JRE as default JRE in Eclipse.

To run the application Net Beans developer can use the maven exec goal but the run configurations for Eclipse developers have to be adapted: Right-click on the ANGERONA App project and select Run as $\rightarrow$ Java Application. As main class select the class *com.github.angerona.fw.app.GUIApplication*. After running the application inspect the Eclipse console for the outputs of the application. There are exceptions saying that some files cannot be found.
Actually the executable runs in the wrong directory. An adaptation of the run configuration fixes the problem. Select *Run* $\rightarrow$ *Run Configurations...* from the menu. Select the ANGERONA App run configuration and switch to the tab *arguments*. Change the working directory to the workspace project of ANGERONA App: *app/target*. Rerun the application and the GUI shall show up without throwing any exceptions. The following list describes the projects of ANGERONA shortly.

- **ANGERONA (Framework)**:
  The framework of ANGERONA consist of all the base classes and the plug-in interfaces. It also implements the functionality of an environment and the agent cycle.

- **ANGERONA (Secrecy Agent Plug-in)**:
  This project contains all operator base classes that define the operation type for the secercy agent model and also contains the implementation of the *Secrecy Knowledge* data component.

- **ANGERONA (Example Plug-in)**:
  A plug-in project with a default implementation of the operators of the secrecy agent model and a simple knowledge representation mechanism that uses a set of propositional formulas.

- **ANGERONA (ASP Plug-in)**:
  A plug-in providing an answer set knowledge representation mechanism. It allows to use different ASP solvers and implements reasoner and revision operators.

- **ANGERONA (Conditional)**:
  A plug-in providing conditionals as knowledge representation mechanism. It uses OCFs for the reasoning.

- **ANGERONA (GUI Extension)**:
  A library extending the ANGERONA framework with UI utility classes. It uses the docking frames library to allow custom perspectives for comparing agent belief bases for example. It also provides an interface for UI-Plug-ins allowing the plug-ins of the ANGERONA framework to define their own UI-Components.

- **ANGERONA (App)**:
  An application using the ANGERONA framework and the ANGERONA GUI extension to give a user the ability to perform experiments using different simulations with different agent and knowledge base representation concepts.

- **ANGERONA (Know-how)**:
  This plug-in implements a planner for the secercy agent model that internally uses the concept of know-how [3].

- **ANGERONA (maven-build-all)**:
  A meta project invoking the goals on all Angerona projects. To run all unit tests in any project of the angerona framework invoke Maven on this project using the commandline: **mvn test** in the root folder of the project (containing the pom.xml) or use *Run as* $\rightarrow$ *Maven build...*. Type "test" into the goal textfield and click onto run.

The following list describes the dependencies of the ANGERONA framework:

- **JSPF**:
  A library implementing a framework for dynamic plug-in loading, the plug-in hierarchy of ANGERONA implements interfaces defined in this plug-in

- **Logback**:
  A library for logging purposes. See this Website for further details

- **Docking Frames**:
  A library which supports a docking behavior for its widgets. Those docking behaviors are very popular in IDEs like Eclipse or Visual Studio.

# Chapter 2

# Learn to develop Plug-ins for ANGERONA

## 2.1 How-to - Create a Custom Plug-in

Now ANGERONA shall start from within the IDE, either Eclipse or Net Beans, such that this Section can start to give an overview over what must be done to implement another agent behavior or a alternative knowledge representation mechanism. The source code of the labeled classes in this Section gives important insights and it is highly recommended to also read the ANGERONA framework chapter of the technical report [1] to get a deeper understanding of the concepts in the ANGERONA framework before extending its functionality. The interface every plug-in needs to implement is *AngeronaPlugin*, alternatively a plug-in class can extend the class *AngeronaPluginAdapter*. This plug-in allows to extend the ANGERONA framework on four different places:

1. An agent component can extend the agent data storage to contain further information like desires, a plan or motivations.
2. An environment behavior allows the ANGERONA framework to change the simulation algorithm, such a more sophisticated behavior could communicate with external simulation software etc.
3. A belief base implementation consists of several parts, the most important part is the belief base itself. But every belief base also needs a translator, a reasoner and a change-operation, like an expansion or a revision. When implementing a new knowledge representation mechanism the developer has to implement at least one of each of those modules.
4. An functional operator, allowing to change the behavior of one of the agents operators, the type of the operators depends on the choosen agent

model. Currently the ANGERONA framework only implements the secrecy agent model described in [2, 1], such that possible operators are *generate-options*, *intention-update*, *subgoal-generation* and *violates*.

We assume that the developer has decided to implement a new agent component to give it's agent model the ability to save the actions that are performed by the agents in the past simulation. Therefore a new maven project is created with the artifact name *new-stuff*, the group-id *angerona.fw* and version *1.0.0*. A component implementing an action history already exists: *angerona.fw.ActionHistory* and Listing 2.1 shows parts of it's code. The developer also has to create a new plug-in class or extend an existing.

Listing 2.1: Code For Action-History Component

```
        /**
         * map saving the action history by storing one list of
         * actions for every agent.
         */
        private Map<String, List<Action>> history = new HashMap<>();

        /** Default Ctor: Used for dynamic creation, creates empty history */
        public ActionHistory() {}

        /** Copy Ctor */
        public ActionHistory(ActionHistory other) {...}

        public void putAction(String agent, Action action) {...}
        public boolean didAction(String agent, Action action) {...}
        public List<Action> getActionsOf(String agent) {...}

        @Override
        public void updateBeliefs(Perception percept, Beliefs oldBeliefs,
                Beliefs newBeliefs) {
                if(percept instanceof Action) {
                        Action act = (Action)percept;
                        putAction(act.getSenderId(), act);
                }
        }

        @Override
        public ActionHistory clone() {
                return new ActionHistory(this);
        }
}
```

The basic idea of a plug-in class is shown in Listing 2.2 and can be seen in *angerona.fw.def.FrameworkPlugin* for example. The annotation *@PluginImplementation* is used to mark those classes that shall be loaded by JSPF. ANGERONA uses lists of Class descriptions to dynamically instantiate those classes, such that a default constructor, that means a parameterless constructor, must be defined. It is also very important that the belief bases and the agent components must support a copy ctor and a clone method because these methods are used internally to perform mental steps. The functional operators are stateless, that means they are pure strategies and have no data, such as class attributes. Classes that are stateless can easily be used in a multi-threaded environment.

Listing 2.2: A JSPF Plugin

```
@PluginImplementation
public class NewPlugin extends AngeronaPluginAdapter {

        @Override
        public List<Class<? extends AgentComponent>> getAgentComponentImpl() {
                List<Class<? extends AgentComponent>> reval =
                        new LinkedList<Class<? extends AgentComponent>>();
                reval.add(ActionHistory.class);
                return reval;
        }
}
```

After the developer has implemented the extension he/she has to adapt configuration files to assure that those implementation got loaded in the simulation. First of all the global configuration file at:
*angerona-dir/software/app/src/main/config/configuration.xml* is used to point to external jar files that contain plug-ins. Adding:

```
<plugin>plugins/new-stuff-1.0.0.jar</plugin>
```

to the *configuration.xml* is only half of the job. The pom.xml files also need an adaption to dynamically copy the plugin jar into the plug-in folder. Therefore take a glance at the section with the dependencies of the file:
*angerona-dir/software/app/pom.xml* for the above defined artifact and group id the following dependency must be added to the file

```
<dependency>
  <groupId>angerona.fw</groupId>
  <artifactId>new-stuff</artifactId>
  <version>1.0.0</version>
  <scope>runtime</scope>
</dependency>
```

Now the build process copies the new version of the new-stuff artifact into the plug-in folder. The next step is to adapt the more specific configuration files. They contain the full java class names of several components, such that they allow to use alternative implementations and add new behaviors to the agent. The agent configuration XML files can be found in the folder: *angerona-dir/software/app/src/main/config/agents*. Those files define the functional operators of the agent, they reference to a script that defines the cycle method of the agent and they contain a collection of agent data components, that are implemented by the agent.

```
<agent-configuration>
  ...
  <operation-set operation-type="GenerateOptions">
    <default-operator-cls>
```

```
      angerona.fw.example.operators.GenerateOptionsOperator
    </default-operator-cls>
    <operator-cls>
      angerona.fw.new.AlternativeGenerateOptions
    </operator-cls>
  </operation-set>
  ...
</agent-configuration>
```

The operation-set defined in the above Listing is used to define default operators and alternative operators for an operation type. In the agent configuration types the operation type must be defined. There must be one *default-operator-cls* element and an arbitrary count of *operator-cls* elements.

The simulation template files are more complex and references an agent configuration file for each agent and for each of the agent's belief bases a belief base configuration file.

```
<simulation-configuration>
  ...
  <behavior>angerona.fw.def.DefaultBehavior</behavior>
  ...
</simulation-configuration>
```

In the simulation template files the behavior element can be used to change the used environment behavior of the simulation. The belief base configuration file is the last type of configuration file, they can be found in the folder: *angerona-dir/software/app/src/main/config/beliefbases*

```
<beliefbase-configuration>
   <name>new belief base</name>
   <reasoners>...</reasoners>
   <change-operators>...</change-operators>
   <translators>...</translators>

   <beliefbase-class>
   angerona.fw.logic.new.NewBeliefbase
   </beliefbase-class>
</beliefbase-configuration>
```

The above listing shows a belief base configuration file. It contains the elements *reasoners*, *change-operators* and *translators*. These elements are operation types and therefore have at least one sub element *default-operator-cls* and an arbitrary count of *operator-cls* sub elements, such that the name of the elements describe the operation type: reasoning-operations, change-operations, translate-operations. But the most important element is *beliefbase-class* which contains the full java class name of the beliefbase implementation that uses the prior defined operations. The Appendix of the technical report contains

a more sophisticated description of the file formats used in the ANGERONA framework.

# Bibliography

[1] Patrick Krümpelmann and Tim Janus. Angerona - A multiagent framework for logic based agents with application to secrecy preservation - draft. Technical report, Technische Universität Dortmund, Department of Computer Science, 2012.

[2] Patrick Krümpelmann and Gabriele Kern-Isberner. Secrecy preserving bdi agents based on answerset programming. In *Proceedings of the 11th German Conference on Multi-Agent System Technologies (MATES'13)*, volume to appear of *Lecture Notes in Computer Science*. Springer, 2013.

[3] Matthias Thimm and Patrick Krümpelmann. Know-how for motivated BDI agents (extended version). Technical Report 822, Technische Universität Dortmund, Department of Computer Science, February 2009.