# REPAST-JADEX PLUGIN USERGUIDE

Version 0.1

20 December 2013

RMIT Agents Group

This tutorial is based on Repast Simphony 2.1 and Jadex 2.3

## Contents

## 1 Introduction

Repast Simphony is a widely used, open source, Agent-Based Modelling (ABM) and simulation platform which is widely accepted by developers who program intelligent agents. Jadex is an open-source platform for developing rich cognitive agents in the Belief-Desire-Intention (BDI) tradition. This document describes how individual agents with sufficiently rich behaviours can be developed using the Repast-Jadex plugin.

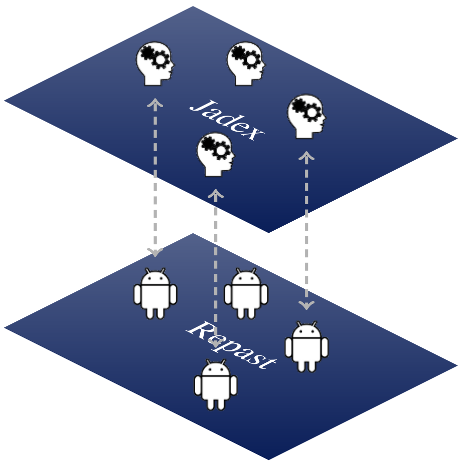


Figure 1: Overview of the Repast-Jadex simulation

In this tutorial, we will extend the Repast Zombies and Human tutorial such that the "brains" of the agents are implemented in Jadex and the "body" of the agents is implemented in Repast. The application works by alternating control between Jadex and Repast on each time step. Reasoning performed in the Jadex brain decides what "actions" the Repast body should do. The Repast body executes these actions in the environment and passes back "percepts" that capture relevant changes that have occurred in the environment. These will again be used by the Jadex brain to make further decisions about what actions the agent body should take next.

# 2 Installation

The following section contains installation instruction for Linux(**??**), Windows(**??**) and MacOS X(**??**).

## 2.1 For Linux

### 2.1.1 Java 7 (JDK7)

Java 7 is a requirement for Repast Simphony 2.1 and Jadex 2.3 which are the main software components for this tutorial.

To check Java version, open Terminal, type and run the following command:

```
java -version
```

To install Java 7, run the following commands:

```
sudo apt-get install openjdk-7-jdk
```

### 2.1.2 Repast Simphony 2.1

Download and Installation instructions of Repast Simphony 2.1 for Linux are based from:
`http://repast.sourceforge.net/download.html`

- First of all you need to download Eclipse Standard 4.3 known as "Kepler" from:
  `http://www.eclipse.org/downloads/`
- Once download is complete, extract the files and copy into:

  /home/NAME/eclipse
- Now go to eclipse folder and run it using following commands:
  ```
  cd /home/NAME/eclipse
  ./eclipse
  ```
- For Linux or Unix, http://repast.sourceforge.net/download.html asks users to install the following additional components via Eclipse Update Manager:

  1. From the default Kepler (Eclipse 4.3) update site:
     `http://download.eclipse.org/releases/kepler`

| Software Component | Category |
|---|---|
| – GMF Tooling 3.1.0<br>– GMF Tooling Runtime Extensions 3.1.0<br>– GMF Tooling SDK<br>– Xpand SDK<br>– Xtend SDK | Modeling |
| – Eclipse XML Editors and Tools | Programming Languages |

**NOTE:** Refer Appendix **??** for steps to install new software in eclipse.

2. From the Groovy Update Site:

   `http://dist.springsource.org/release/GRECLIPSE/e4.3/`

   – Extra Groovy Compilers
   – Groovy-Eclipse

   **NOTE:** Refer Appendix **??** for steps to install new software in eclipse.

- Once the above mentioned additional components are installed, refer Appendix **??** for steps to install new software in eclipse and install Repast Simphony 2.1 using site:

  `http://mirror.anl.gov/repastsimphony`

### 2.1.3 Jadex 2.3

- Download Jadex 2.3 from:

  `http://www.activecomponents.org/bin/view/Download/Overview`

- Once download is complete, extract the files and copy into:

  /home/NAME/jadex-2.3

### 2.1.4 git

Git will be used to clone/download project from bitbucket:

```
sudo apt-get install git
```

### 2.1.5 ant

Ant will be used to run build.xml file for the tutorial.

```
sudo apt-get install ant
```

## 2.2 For Windows

### 2.2.1 Java 7 (JDK7)

Java 7 is a requirement for Repast Simphony 2.1 and Jadex 2.3 which are the main software components for this tutorial. Download and Install Java from: `http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html`

### 2.2.2 Repast Simphony 2.1

Follow instructions from: `http://repast.sourceforge.net/download.html` to install Repast Simphony 2.1 on your Windows machine.

### 2.2.3 Jadex 2.3

- Download Jadex 2.3 from:

  `http://www.activecomponents.org/bin/view/Download/Overview`
- Once download is complete, extract the files into folder named "jadex-2.3" (without quotes).

### 2.2.4 git

Git will be used to clone/download project from bitbucket. Download from: `http://git-scm.com/downloads`

And follow installation instructions from: `https://help.github.com/articles/set-up-git#platform-windows`

### 2.2.5 ant

Ant will be used to run build.xml file for the tutorial. Download .zip file for Windows from: `http://ant.apache.org/bindownload.cgi`

Once the download is complete, follow installation instruction from: `http://madhukaudantha.blogspot.com.au/2010/06/installing-ant-for-windows-7.html`

## 2.3 For MacOS X

### 2.3.1 Java 7 (JDK7)

Java 7 is a requirement for Repast Simphony 2.1 and Jadex 2.3 which are the main software components for this tutorial. Download and Install Java from: `http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html`

### 2.3.2 Repast Simphony 2.1

Follow instructions from: `http://repast.sourceforge.net/download.html` to install Repast Simphony 2.1 on your MacOS X machine.

### 2.3.3 Jadex 2.3

- Download Jadex 2.3 from:

  `http://www.activecomponents.org/bin/view/Download/Overview`
- Once download is complete, extract the files into folder name "jadex-2.3" (without quotes).

### 2.3.4 git

Git will be used to clone/download project from bitbucket. Download from: `http://git-scm.com/downloads`

Now open a Terminal and type in: `git`

It will ask you to install `Xcode` and once that it finish follow installation instructions from: `https://help.github.com/articles/set-up-git#platform-mac`

### 2.3.5 ant

Ant will be used to run build.xml file for the tutorial. Download .tar.gz file for MacOS X from: `http://ant.apache.org/bindownload.cgi`

Open Terminal and follow installation instruction:

- `cd ~/Downloads`
- `tar -xvzf apache-ant-`*`version`*`.tar.gz`
- `sudo mkdir -p /usr/local`
- `sudo cp -rf apache-ant-`*`version`*` /usr/local/ant`
- `export PATH=/usr/local/ant/bin:"$PATH"`
- `echo 'export PATH=/usr/local/ant/bin:"$PATH"' >> ~/.profile`
- `ant -version`

# 3 Pre-requisites

This section describes all the basic steps that will be assumed by the tutorial.

## 3.1 Repast Simphony 2.1

Make sure Repast Simphony 2.1 is running on your machine. This tutorial assumes that you have completed the Repast Java Getting Started tutorial: (`http://repast.sourceforge.net/docs/RepastJavaGettingStarted.pdf`) that steps through the creation of a Human and Zombies application.

## 3.2 Jadex 2.3

Make sure that you can run Jadex. This tutorial assumes you are familiar with the Belief Desire Intention (BDI) agent programming in Jadex and have completed the BDI tutorial: (`http://www.activecomponents.org/bin/view/BDI+Tutorial/01+Introduction`)

## 3.3 Repast demonstration models

Download Repast demonstration models from: (`http://sourceforge.net/projects/repast/files/Repast%20Simphony/Repast%20Simphony%202.0/Repast-Simphony-demos-2.0.zip/download`)

## 3.4 Repast Jadex Plugin packages

Download Repast Jadex Plugin packages from:

(`https://bitbucket.org/dhixsingh/repast-jadex-plugin/downloads`)

**NOTE**: Contact us to access the bitbucket link.

You will need the following files:

1. repast-jadex-common.jar
2. repast-jadex-dev.jar
3. repast-jadex-plugin.jar

You can also build the jars yourself using Terminal for Linux and MacOS X (or git command prompt for Windows) as follows:

- `git clone git@bitbucket.com/dhixsingh/repast-jadex-plugin.git`

  **NOTE**: Contact us to access the bitbucket link.

- `cd repast-jadex-plugin`
- `gedit build.xml`
- Make following changes

  ```
  <!-- CHANGE THIS BASED ON YOUR OWN SETUP -->
      <property name="jadex.dir" value="../../software/jadex"/>
      <property name="repast.dir" value="../../software/
      eclipse37/plugins"/>
  <!-- CHANGE THIS BASED ON YOUR OWN SETUP -->
  ```

  Example like this:

  ```
  <property name="jadex.dir" value="/home/NAME/jadex-2.3"/>
  <property name="repast.dir" value="/home/NAME/eclipse/plugins"/>
  ```

  Save changes in build.xml file and close it, to get back to Terminal.

  **NOTE**: For Windows, the path for example would be: `C:\\User\NAME\jadex-2.3`
- `ant dist`

# 4 Repast-Jadex Tutorial

**NOTE:** The jar files for this tutorial are available in the repast-jadex-plugin folder and other files are available in repast-jadex-plugin/examples/jadex-zombies folder.

## 4.1 Running an existing Repast-Jadex Zomnies application

In this tutorial we will import, setup and run ans existing Zombies application.

The steps for running an existing Zombies application are:

1. Import the Zombies application in Eclipse:
   (a) File → Import Repast Examples
   (b) Select root directory as: "repast-jadex-plugin/examples/jadex-zombies"
   (c) Click Finish.
2. Setup the Zombies application:
   (a) Under the ./lib directory, add the following two Repast-Jadex Plugin jars that are needed for Repast development:
      repast-jadex-common.jar
      repast-jadex-plugin.jar
   (b) Under the ./dep directory, add the following two Repast-Jadex Plugin jar that is needed for Jadex development:
   repast-jadex-dev.jar
3. Run the example as follows:
   (a) First run [Run jadex-zombies brain] and wait till it is read and waiting for a connection from Repast (the body).
   (b) Now run [Run jadex-zombies body] and continue to initialise and step/run as normal.
4. Expected Output

   The output for this tutorial is expected to be similar as to that obtained for "Repast Java Getting Started tutorial" mentioned in section **??**.

## 4.2 Building the Zombies application from parts

In this tutorial we will create a single Eclipse project for developing both the Jadex and the Repast implementation of the Zombies application.

The steps for building the Zombies application are:

1. Create a new Repast project using the project wizard in Eclipse and select Repast Simphony Project. Name the project “jadex-zombies” and click Finish.

2. In the created project, delete the contents of ./src folder that was automatically created with Groovy-related files as we will not use Groovy in this tutorial.

   **NOTE:** Do not delete ./src folder.

3. Under the ./lib directory, add the following two Repast-Jadex Plugin jars that are needed for Repast development:

   repast-jadex-common.jar
   repast-jadex-plugin.jar

4. Create a new folder inside the project (right-click on project and select New→Folder) called dep.

   Add repast-jadex-dev.jar file which is needed for Jadex development.

5. Also add all three Repast-Jadex plugin jars (i.e., repast-jadex-common.jar, repast-jadex-plugin.jar, and repast-jadex-dev.jar) to your project build path.

   To do this, right-click on jar files and Select ”Add to Build Path”

6. Add the Jadex jars to your project build path.

   Under Project Properties do,
   Java Build Path →
   Libraries →
   Add Library... →
   Select User Library then Next →
   User Libraries... →
   New... then create one called ’Jadex’ (without quotes) →
   Add external JARs... then select all the jars in jadex-2.3/lib/*.jar →
   then OK to all the open windows.

7. Copy all files and folders from repast-jadex-plugin/examples/jadex-zombies/src folder into ./src folder of your project.

   The description of folders present are as follows:

| | |
|---|---|
| **brain folder** | It holds all the Jadex reasoning related java classes. |
| **body folder** | It implements the bodily functions of the agent, such as perceiving relevant information in the Repast environment to send to the brain, as well as executing actions requested by the brain. |
| **shared folder** | It stores common information required by both the brain and body, such as what different actions and percepts will be called. |

8. In the **shared** folder, the short description of files is as follows:

| | |
|---|---|
| **HumanLocationTuple.java** | used to record the location of known humans |
| **IdDistanceTuple.java** | uses to record how far the known humans are |
| **Location.java** | used to record the X,Y location of a human |
| **ActionsInterface.java** | contains string names of actions this zombie can perform |
| **PerceptsInterface.java** | contains string names of percepts this zombie can collect |

9. In the **brain** folder, the files present are:

| | |
|---|---|
| **ChangeTarget.capability.xml**<br>**Chase.capability.xml**<br>**Find.capability.xml**<br>**RespondRequest.capability.xml** | Capability Files |
| **Zombie.agent.xml** | Agent Definition File |

The subfolders present in brain folder are:

| | |
|---|---|
| **assesschange**<br>**chase**<br>**find**<br>**respondrequest** | Contains 'Plans' corresponding to specific Capabilities |
| **util** | Contains Tools.java which is used to find nearest Human |

10. In the **body** folder, the short description of files is as follows:

| | |
|---|---|
| **Human.java** | this is the same class that existed in the original tutorial |
| **JZombiesBuilder.java** | context builder with added functionality for Jadex communication |
| **Zombie.java** | extends the BDIAgent class and implements percepts and actions |

11. At the root level, create a jadex-zombies.policy file and add the following text to it:

```
grant {
 permission java.security.AllPermission;
};
```

12. Configure location of Jadex and Repast servers for communication as we use Java RMI for communication between Repast and Jadex. At the root level, create a file ./BDIConfig/BDIConfig.xml with the following contents:

```
<?xml version="1.0" encoding="UTF-8" ?>
<parameters>
 <parameter name="sc_port" defaultValue="1099"/>
 <parameter name="sc_host" defaultValue="localhost"/>
 <parameter name="sc_name" defaultValue="super2"/>
 <parameter name="abm_port" defaultValue="1099"/>
 <parameter name="abm_host" defaultValue="localhost"/>
 <parameter name="abm_name" defaultValue="ABMServerK"/>
</parameters>
```

`sc_host`, `sc_port` and `sc_name` specify BDI server interface address, ie., Jadex. `abm_host`, `abm_port` and `abm_name` specify ABM server interface address, ie., Repast.

`sc_port` and `abm_port` should be same in order to run this tutorial successfully.

`sc_name` and `abm_name` are used by Java RMI (Remote Method Invocation) to find BDI server and ABM server respectively in the registry.

13. Ensure that [Project]→[Build Automatically] is selected.

    At this stage all the project files should be built and there should not be any compile errors.

14. Copy the following files from the existing Repast JZombies_Demo project into the jadex-zombies.rs/ sub-directory, overwriting any existing files:

    context.xml
    parameters.xml
    repast.simphony.action.data_set_1.xml
    repast.simphony.action.display_3.xml
    repast.simphony.action.file_sink_2.xml
    repast.simphony.action.time_series_chart_4.xml
    repast.simphony.dataLoader.engine.ClassNameDataLoaderAction_0.xml
    scenario.xml
    styles/

15. Copy the following files from the existing Repast JZombies_Demo project into the icons/ sub-directory:

    person.png
    zombie.png

16. At the root level, put the jadex-zombies.properties file and make sure the properties are set as follows:

```
build_location=./bin/brain/Zombie.application.xml
package=brain
Zombie=./bin/brain/Zombie.agent.xml
```

| Attribute | Description |
|---|---|
| **build_location =** | Speciy the build location of application definition file. |
| **package =** | Speciy the package where the application definition file belongs. |
| **Zombie =** | Specify agent definition file location. The required format is: AGENT_REPAST_CLASSNAME = JADEX_AGENT_XML_PATH |

17. We will now create separate run configurations for the brain and body.

    (a) In run configurations, select [jadex-zombies Model] and rename it to [Run jadex-zombies body]. Ensure that is has the following setup:

    **Project:** jadex-zombies

    **Main class:**
    repast.simphony.runtime.RepastMain

    **Program arguments:**
    "$workspace_loc:jadex-zombies/jadex-zombies.rs"

    **VM arguments:**
    -Xss10M -Xmx400M -Djava.security.policy=jadex-zombies.policy

    **Classpath:**
    Repast Simphony Interactive Runtime Libraries
    Jadex (User Library)
    dep/repast-jadex-dev.jar
    lib/repast-jadex-common.jar

    **NOTE:** The Jadex libraries and repast-jadex-dev.jar are actually not needed for the body part of the program. However, the Repast runtime class loader looks for classes referenced in the the project sources which includes the brains side of things. So these are needed just to keep the Repast runtime class loader happy. If the Jadex development (brains) was done in a separate project, this would not be necessary.

    (b) Right-click the [Run jadex-zombies body] configuration and select Duplicate. Rename this duplicate to [Run jadex-zombies brain]. Edit the configuration with the following setting:

    **Project:** jadex-zombies

    **Main class:**
    rmit.agent.jadex.abm_interface.super_central.ABMSimStarter

    **Program arguments:**
    -prop jadex-zombies.properties -console -f -threaddefer false
    `Refer Appendix` **??** `for explanation of Program arguments`

    **VM arguments:**
    -Xss10M -Xmx400M -Djava.security.policy=jadex-zombies.policy

    **Classpath:**
    Repast Simphony Interactive Runtime Libraries
    Jadex (User Library)
    dep/repast-jadex-dev.jar
    lib/repast-jadex-common.jar
    jadex-zombies (Project)

18. Now you can run the example as follows:

    (a) First run [Run jadex-zombies brain] and wait till it is read and waiting for a connection from Repast (the body).

    (b) Now run [Run jadex-zombies body] and continue to initialise and step/run as normal.

19. Expected Output:

    The output for this tutorial is expected to be similar as to that obtained for “Repast Java Getting Started tutorial” mentioned in section **??**.

### 4.3 Understanding the Repast-Jadex application

#### 4.3.1 Repast Implementation

The **Repast Implementation** is present in **body** folder and contains following files:

Human.java
JZombiesBuilder.java
Zombie.java

---

The *JZombiesBuilder* class shown in Listing **??** extends BDIAbstractContextBuilder which is an abstract class containing methods required to create a BDI context. The method implemented in *JZombiesBuilder* class is *BDIbuild* which is used to create *Context*. The understanding of *JZombiesBuilder* class is same as that of *ContextBuilder* class used in "Repast Java Getting Started Tutorial" referred in Section **??**.

Listing 1: JZombieBuilder class

```
public class JZombiesBuilder extends BDIAbstractContextBuilder {
    @Override
    public Context<Object> BDIbuild(Context<Object> context) {
        context.setId("jzombies");
        ...
        return context ;
    }
}
```

---

In order to create BDI agents, Zombie class extends BDIAgent class. All the abstract methods present in BDIAgent class are implemented in Zombie class.

```
public class Zombie extends BDIAgent
```

The three methods discussed are: `getPercepts()`, `queryPercept(String perceptID)` and `doAction(String actionID, ActionContent actionContent)`.

---

The `getPercepts()` method is as shown in Listing **??**:

Listing 2: getPercepts method

```
public class Zombie extends BDIAgent implements PerceptsInterface,
ActionsInterface {
    ...
    public PerceptContainer getPercepts() {
        lookForHumans();
        updateMyLocation();
        return perceptContainer;
    }
    private void lookForHumans() {
        ArrayList<HumanLocationTuple> nghHumans = new
        ArrayList<HumanLocationTuple>();
        GridPoint pt = grid.getLocation(this);
        GridCellNgh<Human> nghCreator = new
        GridCellNgh<Human>(grid, pt, Human.class,
        neighboring_view_range, neighboring_view_range);
        List<GridCell<Human>>
        gridCells = nghCreator.getNeighborhood(true);
        for (GridCell<Human> cell : gridCells) {
            if (cell.size() > 0) {
                Iterable<Human> humans = cell.items();
                for (Human human : humans) {
                    pt = grid.getLocation(human);
                    HumanLocationTuple newHuman = new
                    HumanLocationTuple(human.toString(),
                    new Location(pt.getX(), pt.getY(), 0));

                    if (nghHumans.contains(newHuman) == false)
                    nghHumans.add(newHuman);
                }
            }
        }
        setPercept(NEIGHBOURING_HUMANS, nghHumans);
        super.setNewChange(true);
    }
    public void updateMyLocation()
    {
        GridPoint myLocation = grid.getLocation(this);
        setPercept(MY_LOCATION, new Location(myLocation.getX(),
        myLocation.getY(), 0));
        super.setNewChange(true);
    }
    ...
}
```

Method `getPercepts()` returns all the percepts collected by the agent during the last interval. As you can see in `getPercepts()` code, it calls `lookForHumans()` and `updateMyLocation()` methods.

The code use `setPercept(String Percept_ID, Object value)` to add a new percept to a member variable defined in the super class called `perceptContainer`. We simply return the latter member variable to the caller method.

In the provided example, we first look for any changes about the neighboring humans location, if they moved or a new human came to the zombie vision range we add it to a hash-map. This process is coded in method *lookForHumans()*.

As Jadex counterpart can easily deal with Object array but not hash-maps we also convert the return value to an array and then add it to the percept-container. Note that as we established a mechanism to only notify Jadex when there is a new change including both new percepts and action state, you need to use *super.setNewChange(true)* when you want to send your data to Jadex.

We also get the percept information of Zombies location using process defined in method *updateMyLocation()*.

---

The `queryPercept(String perceptID)` method is as shown in Listing **??**:

Listing 3: queryPercept method

```java
@SuppressWarnings("unchecked")
@Override
public Object queryPercept(String perceptID) {
    if (perceptID.equalsIgnoreCase(FELLOW_ZOMBIE_LIST)) {
        Context<Zombie> context = ContextUtils.getContext(this);
        IndexedIterable<Zombie>
        agents = context.getObjects(Zombie.class);
        HashMap<String, Location>
        zombieLst = new HashMap<String, Location>();
        for (Zombie agent : agents) {
            if (agent != this) {
                GridPoint pt = grid.getLocation(agent);
                zombieLst.put(agent.toString(),
                new Location(pt.getX(), pt.getY(), 0));
            }
        } return zombieLst;
    } return null;
}
```

Method `queryPercept(String perceptID)` is designed to send ad hoc queries. You need to handle perceptID and return a proper *Object*.In `queryPercept(String perceptID)`, if the caller method request for a list of zombies, the callee returns list of all other zombies in the context. Any other queries return *null*.

---

The `doAction(String actionID, ActionContent actionContent)` method is an interface to ask BDI agents to perform a specific action in the next interval.

The MOVE_TOWARDS code section from the doAction() method is as shown in Listing **??**:

Listing 4: MOVE_TOWARDS action from doAction method

```
public ActionContent.State doAction(String actionID,
ActionContent actionContent) throws Exception {
    ...
    if (actionID.equalsIgnoreCase(MOVE_TOWARDS)) {
        @SuppressWarnings("unchecked")
        Context<Human> context = ContextUtils.getContext(this);
        IndexedIterable<Human>
        agents = context.getObjects(Human.class);
        String humanID = (String) actionContent.getParameters()[0];
        for (Human agent : agents) {
            if (humanID.equalsIgnoreCase(agent.toString())) {
                moveTowards(grid.getLocation(agent));
                if (infect()) {
                    tryNumber = 0;
                    super.setNewChange(true);
                    return ActionContent.State.PASSED;
                }
                if (++tryNumber >= max_try_number) {
                    tryNumber = 0;
                    super.setNewChange(true);
                    return ActionContent.State.FAILED;
                }
                return ActionContent.State.RUNNING;
            }
        }
        // could not find the human
        tryNumber = 0;
        super.setNewChange(true);
        return ActionContent.State.FAILED;
    }
    throw new Exception("Unknown action:" + actionID);
}
```

In case of MOVE_TOWARDS action, the zombie tries to locate the human. If it could be able to locate the human it takes one step to reach it; otherwise, it returns a failure. If zombie could infect a human during its next step based on a success on locating the human, it returns success; otherwise, if a number of maximum tries reached it returns failure.

### 4.3.2 Jadex Implementation

The **Jadex Implementation** is present in **brain** folder and contains following files and folders:

| **Files** | **Folders** |
|---|---|
| ChangeTarget.capability.xml | assesschange |
| Chase.capability.xml | chase |
| RespondRequesdt.capability.xml | respondrequest |
| Find.capability.xml | find |
| Zombie.agent.xml | util |

The capability files are the XML responses that Repast agent receives when the Repast agent makes a request on the percepts.

1. **ChangeTarget.Capability.xml:** This capability file holds the definition of how is the possibility of Zombie's agent to change target. It can change target if there is a closer Human using `SwitchTarget_NHcloserPlan`, or if the previous target is out of his range of looks using `SwitchTarget_TNotSeenPlan`.

   The files *SwitchTarget_NHcloserPlan.java* and *SwitchTarget_TNotSeenPlan.java* are present in **assesschange** folder.

2. **Chase.Capability.xml:** This capability file holds the definition of how the agent could chase a Human, it is quite simple that it only has one plan which is `ChaseHumanPlan`.

   In each of the `ChaseHumanPlan` revisitiation, an AssesChangeG is fired, it will be taken care by plan inside the *ChangeTarget.capability.xml*.

   The file *ChaseHumanPlan.java* is present in **chase** folder.

3. **Find.capability.xml:** This capability file holds the definition of how the Zombie can find the Human to chase.

   It uses 4 plans present in **find** folder:

   (a) *ChooseNHPlan.java:* Trying to see around and looking for Humans (without moving location).

   (b) *GetFromColleagePlan.java:* In case `ChooseNHPlan` fails, the Zombie will try to look for information from Colleagues.

   (c) *AskColleaguePlan.java:* This plan is called by `GetFromColleagePlan` to send messages and to communicate with its Zombie's Colleagues in order to get information about Humans.

   (d) *SearchPlan.java:* When the Zombie fails to get information using above three plans, it uses this plan to move around and find human by itself.

   All the above four plans are triggered by `FindTargetGoal` which is defined in *Zombie.agent.xml*. `FindTargetGoal` will be created whenever the Zombie believes that it does not have any TargetHuman.

4. **RespondRequest.Capability.xml:** This is the capability of Zombie agent to receive request from its colleagues on the list of human that it knows.

   This capability uses the *ProvideRequestPlan.java* present in **respondrequest** folder.

   The `ProvideRequestPlan` is triggered whenever there is an incoming message from other ZombieAgent (Message defined in *Zombie.agent.xml*). In turns it will reply with a message called HumanListM.

5. **Zombie.agent.xml:** This file defines a Zombie Agent. A Zombie knows the following information:

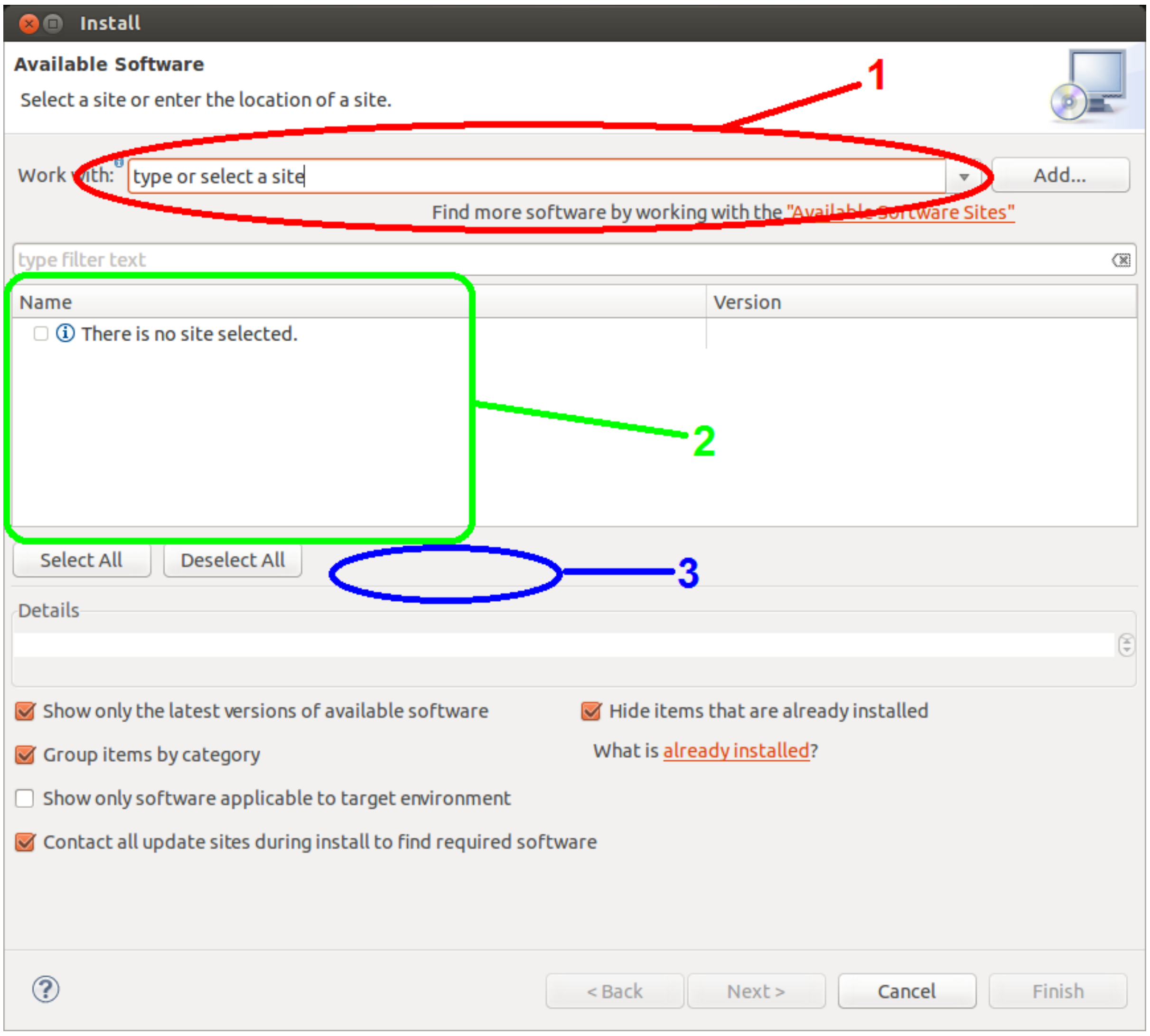
   - List of Human around it
   - The Human it is targeting
   - Its own location

   This file definition becomes the point of connection of the other capabilities (`ChangeTarget, Chase, Find,` and `RespondRequest`).

   It holds the Goals and Messages which are used to glue all the capabilities functionalities.

## A How to Install New Software in Eclipse

1. Help → Install New Software
2. Enter site in textbox next to ’Work with:’ marked as 1 inFigure **??**.



**Available Software**

Select a site or enter the location of a site.

Work with: type or select a site

Find more software by working with the "Available Software Sites"

type filter text

| Name | Version |
|---|---|
| There is no site selected. | |

Details

[✓] Show only the latest versions of available software

[✓] Group items by category

[ ] Show only software applicable to target environment

[✓] Contact all update sites during install to find required software

[✓] Hide items that are already installed

What is already installed?

Figure 2: Steps to Install New Software in Eclipse

3. Now select components from the list that appears in ’Name’ section marked as 2 in Figure **??**.
4. Section marked as 3 in Figure **??**, shows the number of items selected.
5. Click Next, Next.
6. Accept terms and conditions and Click Finish.
7. Once done it will ask to restart.

## B Command line arguments

```
-prop <configuration_file_location>
    This option is to give the information on which configuration
    files is to be used.

-console
    This option is used to show the logging on console.

-f, --force
    This option is used to overwrite the existing CentralOrganizer
    files.
```

# C How to define Agents in Jadex

The agents defined in this section will represent BDI agents in Repast.

## C.1 Create Agent Definition File and Capability File Template

One of the two step in creating agents in Jadex is to make the agent definition file, which is written in XML. In order to equipped Jadex's agents with the abilities to collaborate with Repast, several extra activities are required when building the agent definition file.

### C.1.1 Agent Definition and Capability File Template

Every Agent Definition File or Capability File are required to be created out of a template given in *repast-jadex-dev.jar* package, refer section **??**. The detailed instruction on how to use the template are:

1. Copy the template files and rename it.
   - For Agent Definition File: Create a specific agent definition file based on *repast-jadex-dev/rmit/agent/jadex/abm_interface/agent/Template_AgentDefinition.agent.xml* file and rename it as *zombie.agent.xml*.
   - For Capability File: Create a specific capability file will be based on *repast-jadex-dev/rmit/agent/jadex/abm_interface/agent/Template_Capability.capability.xml* file and rename it as *find.capability.xml*.
2. Inside the definition file (from step 1), go to the start tag of element *agent* (or *capability* for capability file). Change the attributes *name* into your file's name (without *.agent.xml* or *.capability.xml* part), and the attribute *package* into the package where the file is relative to your project.
3. Do not change anything (except assignments to inner capabilities for each belief) inside the file, starting from the comment

   ```
   <!-- System Default Part -->
   ```

   until you reach the part of file commented as

   ```
   <!-- Start Your Specific Agent Definition Here On -->
   ```

   From this part of file, insert the code one usually (default) write for an agent definition file in Jadex. The components of the file before the comment, are necessary for detecting the idle state of an agent, which support the architecture design of Jadex-Repast collaboration.

   **NOTE**: Currently there is a requirement for the user to add assignment of beliefs towards each of user-specific capabilities enclosed in the template comment starting with:

   ```
   <!-- System Default Part -->
   ```

   and ended with :

   ```
   <!-- End Of System Default Part -->
   ```

For example:

```
<belief name="isIdle" class="Boolean">
 <fact>false</fact>
 <assignto ref="InnerCapability.environment"/>
</belief>
```

Users need to adjust the `<InnerCapability>` part (without angle brackets) in the example to their corresponding capability's name, and apply this to the assignment of the whole default beliefs provided for this framework.

### C.1.2 Agent's Percepts

Percepts is the information of the environment that could be perceived by an agent in an Agent Based Model Simulation (ABMS). In terms of Jadex-Repast collaboration, we would like that the percepts of an agent in Repast could be used as basis of information for the corresponding agent represented in Jadex BDI. Therefore there is a need to define this ABMS percepts in Jadex BDI agent.

Based on the frequencies of passing the percepts value information from Repast to Jadex, there are two types of percepts:

- Percepts which are routinely passed to Jadex in every time-step of Repast
- Percepts which are passed to Jadex only when it is required by Jadex. (adhoc Percepts)
  There are percepts in Repast which are not useful and only waste computation resources, when it is updated to Jadex routinely. These are the percepts which do not have the possibilities to trigger any event in Jadex BDI agent, but might become essential information in particular steps of plan processing.

The classification of percepts is application specific. Due to this, percepts are define in two ways for Jadex Agent. Both of them are defined as belief with different belief's content. The following is a detailed description on how to define the percepts:

- For percepts which are routinely passed:
  Define your percepts in agent definition file under the agents `<beliefs>` element as follows:

  `<belief` name="*percept_id*" `class`="*value_type*">
          `<fact>`*value*`</fact>`
  `</belief>`

  The italics words are required to be replace with the respected percepts data. *percept_id* is the id of percepts which is the same with the one used in Repast. *value_type* is the type of value for the percept. *value* is the initial value.

- For adhoc percepts :
  Define your percepts in agent definition file under the agents `<beliefs>` element as follows:

```
<belief name="percept_id" class="value_type" evaluationmode="pull">
        <fact>
            AdhocPerceptQuery.query((String)
            $beliefbase.myself.getProperty("agentID"),
            "percept_id",$beliefbase.environment.getProperty("server"))
        </fact>
</belief>
```

The italics words are required to be replaced with the percepts data define for the specific application. *percept_id* is the id of percepts which is the same with the one used in Repast. *value_type* is the type of value for the percept.

**NOTE**: There is a requirement to relate the percepts in Repast with its representation in Jadex. Thus the *percept_id* defined in Repast should be exactly same as *percept_id* defined in Jadex.

## C.2 Agent's Plans

The second step in creating agents in Jadex is to define the Plans used by the agent in the form of Java code. This section describes steps to create the agent's plans required for collaborating with Repast.

### C.2.1 Basis Class of Agent's Plan

*Plan Classes* are made by extending *repast-jadex-dev/rmit/agent/jadex/abm_interface/agent/StepPlan*. While creating a plan in Jadex, in order to collaborate with Repast, the developer is required to extends `rmit.agent.jadex.abm_interface.agent.StepPlan`, instead of the default class `jadex.bdi.runtime.Plan` provided by Jadex.

### C.2.2 Accesing the Agent's percepts in Plan's procedure

Percepts are information of environment condition which are perceived by agents in Repast. This information is passed to Jadex's agents in order to synchronize the knowledge of Jadex's agent with its representative in Repast. As is mentioned in section **??**, percepts are implemented in Jadex as agent's belief. Therefore, to access the percepts value, the developer needs to do it in the similar way they access ordinary Jadex's belief.

For example: `getBeliefbase().getBelief("percept_id").getFact();`

### C.2.3 Agent's Action

Action is the set of procedure ran by an agent in ABMS which would affect the simulation environment. The Repast-Jadex tutorial, collaborated Repast as the ABMS, with Jadex as the BDI Agent platform. Whenever an agent with BDI reasoning is required in the simulation, the corresponding agent in Repast would have a representative in Jadex.

Jadex would provide the BDI infrastructure as the basis of agent's reasoning. There are times that the result of the reasoning would initiate an action to be done by Repast's agent. Therefore there is a need of methods to interface the communication of actions desired by the reasoning.

In order for the developer to express the point where actions should be initiated in simulation, they would need to call the following function inside the `StepPlan`:

```
void act (String action_id, Object[] parameter);
```

***action_id*** is a unique id of an action determining which action is to be done in simulation. There is a need of equivalencies of `action_id` defined in Repast as well as in Jadex. It is recommended for the developer to create an interface of this `actions_id` in order to guarantee the same id is used in Repast and Jadex.

***parameter*** is the set of arguments desired to be passed for the action. For example, if we have an action of `walkTo` defining where we want to *walkTo* then we might call the `act` method such as :

```
act ("walkTo", "School");
```

## C.3 Application Configuration File

After defining agents involved in the application, one would need to define the configuration file for Jadex's agent. This configuration file is written as java properties file. A template file location: *repast-jadex-dev/rmit/agent/jadex/abm_interface/Template_PropertiesFile.properties*.

The properties file consist of :

1. Location to Save Log Files
   This is the path where all the logging files are saved in. For example:
   `/../test/logging`

   **NOTE**: For Windows, the path for example would be: `C:\\test\logging`

2. SuperCentralOrganizer (SC) server address (host, port, and name)
   In Repast-Jadex tutorial, there should be an abilities to have a distributed application for the simulation which host tons of agents. Therefore we adopt a centralized distributed architecture. The high level architecture of Jadex system would consist of a SuperCentralOrganizer (SC) and multiple CentralOrganizer (CO). Both of them are an instantiation of Jadex application. SC has the role to be an entity who manages and monitors the condition of all the other COs. CO is where the Jadex agents are directly monitored and managed. In order to support the distributed system, it is required to determine the host, port and name of the SC server. For example:

   ```
   sc_host = localhost
   sc_port = 1099
   sc_name = jadexSC
   ```

3. CentralOrganizer (CO) port address

   This defines the port of CO. For example:

```
co_port = 1099
```

4. Build location of application definition file.
   This is the path where you want to put the resulting CO file, which is generated automatically specific to your application and based on the configuration files. For example:
   ```
   /../test/build
   ```
   **NOTE**: For Windows, the path for example would be: `C:\\test\build`

5. The package of application definition file.
   The application definition files refers to CO. The package refers to location of CO. It is in analogy with the attribute `package` under `agent` element's start tag, in Jadex's agent file definition. For example:
   ```
   package = package_name
   ```

6. Maximum and minimum capacities of a CO
   This determines the number of maximum and minimum agents lived in a CO. The purpose of the maximum limitation is to prevent overfill situation of agents in a CO, which could lead to system crash due to the machine is out of computation capacities. Minimum limitation is to prevent the condition where there are only a very small number of agent lived in various CO, which would waste the efficiency when these agent needs to communicate frequently. For example:
   ```
   maxCapacityPerCO = 100
   minCapacityPerCO = 25
   ```

7. List of agent's definition file location
   This is where the first connection on agents in Jadex and Repast is build. The required format is as follows:
   ```
   AGENT_REPAST_CLASSNAME=JADEX_AGENT_XML_PATH
   ```
   The XML file determines the representative of a Repast agent type in Jadex. Developers are required to list all of the possible agent type which are intended to be created in Jadex. For example:
   ```
   Zombie = /../test/zombie.agent.xml
   Human = /../test/human.agent.xml
   ```
   **NOTE**: For Windows, the path for example would be: `C:\\test\zombie.agent.xml`