REPAST-JADEX QUICK START GUIDE

Version 0.1 DRAFT

03 June 2014

RMIT Agents Group

Contents

1 Introduction

Repast Simphony[1] is a widely used, open source, Agent-Based Modelling (ABM) and simulation platform which is widely accepted by developers who program intelligent agents. Jadex[2] is an open-source platform for developing rich cognitive agents in the Belief-Desire-Intention (BDI) tradition[3]. This document is a quick start guide to building agent-based simulations in Repast, using Jadex BDI agents.

Figure 1 shows the overall architecture of a Repast-Jadex simulation built in this way, using the Repast-Jadex plugin framework. The idea is to offload the decision making function of some Repast agents, to the Jadex process, such that essentially their "brains" live in the Jadex system and their "bodies" in Repast. This arrangement still allows for these BDI-enhanced Repast agents to co-exist with regular Repast agents, programmed in the usual way. Similarly, other cognitive agents may exist in the Jadex process that do not have a physical embodiment in the Repast process.

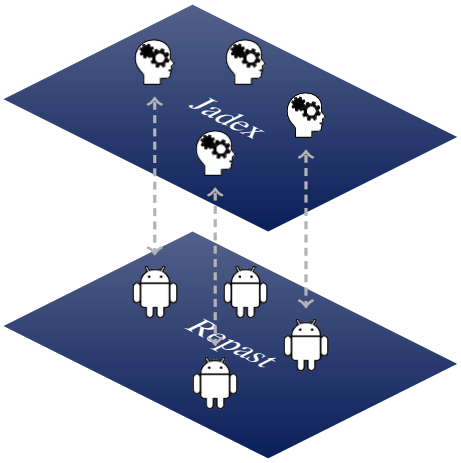

Figure 1: Overview of the Repast-Jadex simulation

The system works by alternating control between Jadex and Repast on each time step[4]. Reasoning performed in the Jadex brains decides what "actions" the Repast bodies should perform. The Repast bodies executes these actions in the environment and passes back the status' of these actions along with "percepts" that capture relevant changes that have occurred in the environment. These will again be used by the Jadex brains to make further decisions about what to do next.

In this tutorial, we will extend the Repast Zombies and Human tutorial[5] to build "smarter" zombies that coordinate their moves. In this new setup, the zombies have limited information about their surroundings (they were after all humans once), and can only see upto a fixed distance around them. If a zombie cannot see any humans in its immediate vicinity, it can ask other zombies about what they see. Those zombies may respond with the locations of any humans they can spot.

[1] `http://repast.sourceforge.net`
[2] `http://sourceforge.net/projects/jadex`
[3] `http://www.activecomponents.org/bin/view/BDI+Tutorial/01+Introduction`
[4] The Repast and Jadex processes communicate using Java RMI.
[5] `http://repast.sourceforge.net/docs/RepastJavaGettingStarted.pdf`

2 Pre-requisites

The instructions in this tutorial assume a unix-based setup, and were tested on a machine running Linux Mint 13 x86. Since Repast and Jadex both support development on other operating systems such as Windows and Mac OS X, these instructions can also be used for other set ups, with some operating system specific modifications.

To get started, you will need the following:

- **Repast Simphony 2.1** (`http://repast.sourceforge.net`)
  This tutorial assumes that you have Repast Simphony 2.1 installed, and have completed the Repast Java Getting Started tutorial[6] that steps through the creation of a Human and Zombies application.

- **Jadex 2.4** (`http://sourceforge.net/projects/jadex`)
  This tutorial assumes that you have Jadex 2.4 installed, and have completed the BDI programming tutorial for Jadex[7].

- **Repast-Jadex Plugin 0.1** (`https://bitbucket.org/dhixsingh/repast-jadex-plugin`)
  Please contact Dhirendra Singh (dhi.singh@gmail.com) for access to this package.
  The distribution contains the following files:

repast-jadex-quickstart.pdf	This quick start guide.
repast-jadex-common.jar	Provides data and interface classes for communication between the Repast and Jadex processes.
repast-jadex-dev.jar	Provides the interface layer for Jadex. This layer provides the infrastructure for unpacking and distributing incoming action status' and percepts to the Jadex agents, as well as collecting actions from the Jadex agents to send to the Repast process.
repast-jadex-plugin.jar	Provides the corresponding interface layer for the Repast process. It provides the infrastructure for unpacking and distributing incoming actions to the Repast agents, as well as collecting status' and percepts to send to the Jadex process.
commons-cli-1.2.jar	Used by the Jadex process for parsing command line arguments.
examples.tgz	Contains the example Jadex Zombies application.

[6] `http://repast.sourceforge.net/docs/RepastJavaGettingStarted.pdf`
[7] `http://www.activecomponents.org/bin/view/BDI+Tutorial/01+Introduction`

3 Getting started with the Jadex Zombies example

The complete Jadex Zombies application introduced in Section 1 is provided in the Repast Jadex Plugin distribution. If you have already installed the pre-requisites (Section 2), then you can be getting up and running with the example application is straightforward:

1. Import the provided Jadex Zombies application into Eclipse:
   (a) File → Import Repast Examples
   (b) Select examples.tgzthen pick the jadex-zombies-demo project
   (c) Click Finish

   At this point you may see some compile errors if Eclipse is setup to Build Automatically. That's ok, we will now fix the project setup.

2. Add the required JARs:
   (a) Under the ./lib directory, add the JARs needed for Repast-side development:
   repast-jadex-common.jar
   repast-jadex-plugin.jar
   (b) Under the ./dep directory, add the JARs needed for Jadex-side development:
   repast-jadex-dev.jar
   commons-cli-1.2.jar
   (c) Add the Jadex distribution JARs as follows. Under Project Properties do:
   Java Build Path →
   Libraries →
   Add Library... →
   Select User Library then Next →
   User Libraries... →
   New... then create one called Jadex →
   Add external JARs... then select all the JARs in JADEX_ROOT/lib/*.jar →
   Click OK on all the open windows.

3. Your Eclipse project should now compile without errors. Now, run the example as follows:
   (a) Launch Run jadex-zombies brain.
   This starts the Jadex process along with the Java RMI registry, and waits for a connection from the Repast process.
   (b) Next, launch Run jadex-zombies body. This launches the Repast GUI. Run the Repast simulation as normal. This will initiate the Java RMI connection to the already running Jadex process, and you should see Zombies and Humans moving around on the familiar landscape[8].

[8] http://repast.sourceforge.net/docs/RepastJavaGettingStarted.pdf

4 Understanding the Jadex Zombies example

4.1 Eclipse Project Setup

Figure 2 shows the structure of how the Jadex Zombies example is setup in Eclipse. Here, complete application development is done within a single Eclipse project. This includes the Jadex reasoning for the Zombies' brains (package brain), the Repast simulation containing the Humans and the Zombies' bodies (package body, as well as the agreed messages such as names of actions sent from Jadex and names of percepts sent from Repast (package shared).[9]

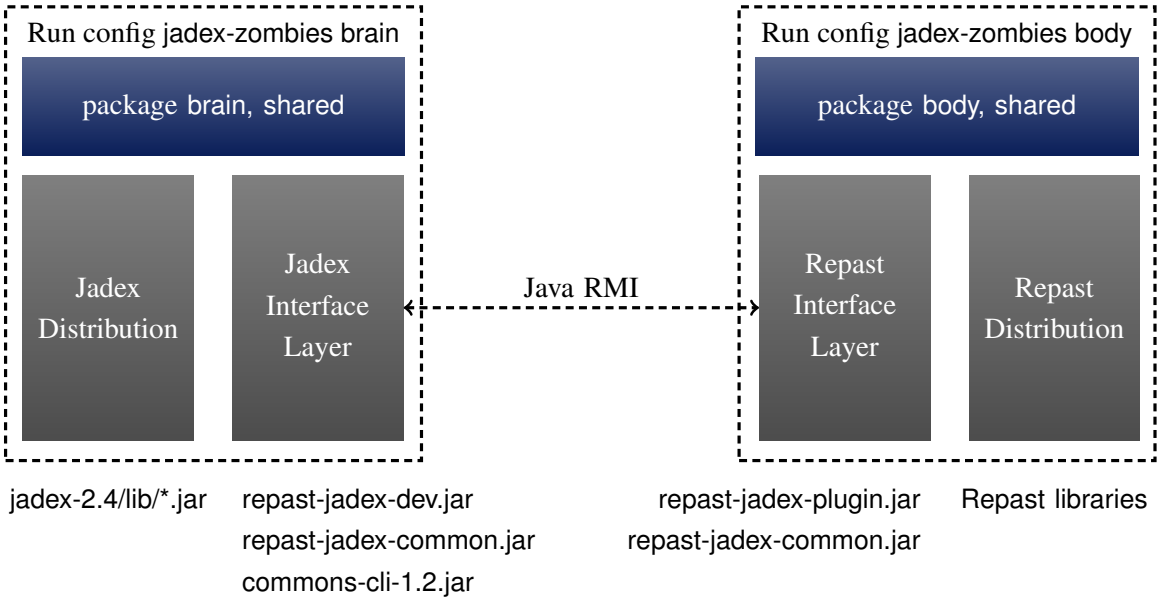

Figure 2: Jadex Zombies application setup

4.2 Troubleshooting Java Classpath issues

If you run into classpath problems, Figure 2 should help you in understanding which packages are required where. Building the Jadex Zombies application should be fairly simple if you have followed the steps in Section 3. Runtime setup is slightly more involved, and is explained below.

Run Configuration jadex-zombies brain:

If you launch this run config and get classpath errors, there are two things to check. First, that your Jadex installation is being picked up correctly as described in Section 3. If you were able to build the example successfully, then this should already be the case. Just make sure that the Jadex User Library is also included in the jadex-zombies brain Run Configuration classpath setup.

Second, that the setup for Java RMI is correct, and that the RMI registry is being started correctly.

[9] An alternative setup could have been to have two separate Eclipse projects, one for the Jadex-side development (with packages brain and shared), and one for the Repast-side (with packages body and shared).

These are configured in the files ./jadex-zombies-demo.properties and jadex-zombies-demo.policy and are passed as command line arguments to the jadex-zombies brain Run Configuration using -prop jadex-zombies-demo.properties and -Djava.security.policy=jadex-zombies-demo.policy.

If everything is configured correctly, you should see the following messages on launch:

```
INFO: Started platform...
INFO: SuperCentralOrganizer created and registered...
INFO: Started Central Organizer component...
INFO: Central Server created and registered...
INFO: Register to rmi://localhost:1099/SuperCentralJadex/super2
INFO: Try to register : Proxy\ldots: isSuccess-true
```

Run Configuration jadex-zombies body:

If you launch this configuration and do not see the Repast GUI, or see the GUI but cannot initalise the simulation, then you have a problem with the Repast runtime configuration. Repast loads all its compiled code and the JARs on which that depends using a plugin classloader. Any code that you write that depends on Simphony code also needs to be loaded using that classloader. This means that you should **NOT**, have repast-jadex-plugin.jar in the classpath setup of the runtime classloader in jadex-zombies body Run Configuration. If you do, you will get ClassNotFoundExceptions relating to Repast classes which have not been loaded yet by the plugin classloader. The correct way to do this is to put repast-jadex-plugin.jar in the ./lib directory, which is searched by the plugin classloader for JARs to load. Moreover, you will want to add an entry `<agents path="../lib"/>` to the file ./jadex-zombies-demo.rs/user_path.xml. This tells Repast to also search the ./lib directory for agents related classes.

If you can successfully initialise the Repast simulation and see the Zombies and Humans on the landscape display, but but classpath errors as soon as you step/run the simulation, then your problem is most likely related to Java RMI, and particularly that the RMI Client (Repast side) cannot find the Server related classes (Jadex side). You can do three things to resolve this. First, check that ./BDIConfig/BDIConfig.xml exists and has the same RMI setup as the Jadex side. Second, ensure that repast-jadex-dev.jarand repast-jadex-common.jar(but not repast-jadex-plugin.jar) are included in the jadex-zombies body Run Configuration. Third, make sure the security policy file is specified at in the VM arguments using -Djava.security.policy=jadex-zombies-demo.policy.

4.3 Repast Implementation

The application code of the Repast side resides in the body package and contains following files:

Human.java
JZombiesBuilder.java
Zombie.java

The *JZombiesBuilder* class shown in Listing 1 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 2:

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 3:

Listing 3: queryPercept method

```
@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 4:

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.

4.4 Jadex Implementation

The application code of the Jadex side resides in the brain package and contains following files and folders. The capability files are the XML responses that Repast agent receives when the Repast agent makes a request on the percepts.

1. assesschange/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/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/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/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. util/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 3.

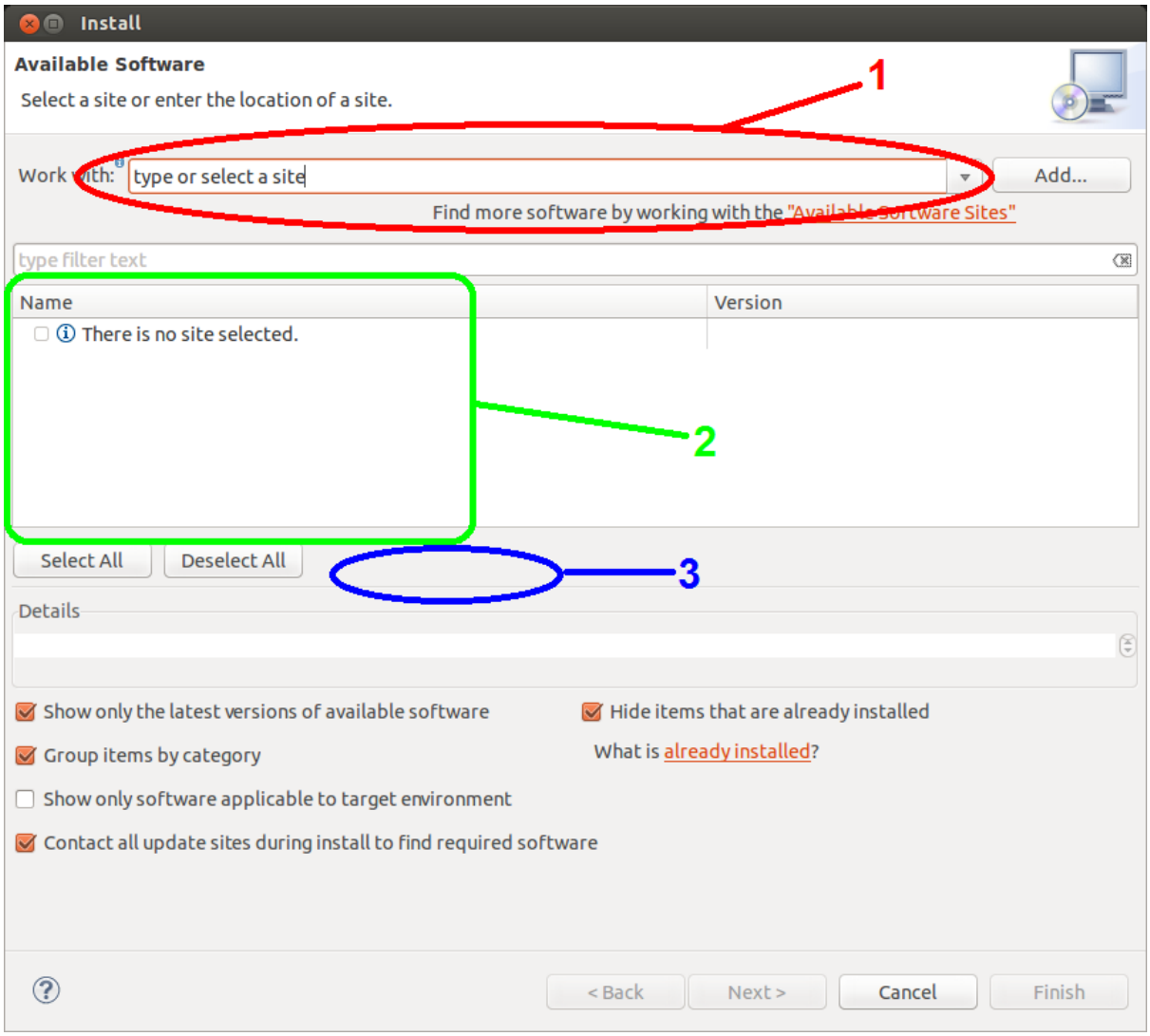

Figure 3: Steps to Install New Software in Eclipse

3. Now select components from the list that appears in 'Name' section marked as 2 in Figure 3.
4. Section marked as 3 in Figure 3, 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 How to define Agents in Jadex

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

B.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.

B.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.

B.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.

B.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.

B.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.

B.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 B.1.2, 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();`

B.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");
```

B.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`

C 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.
```