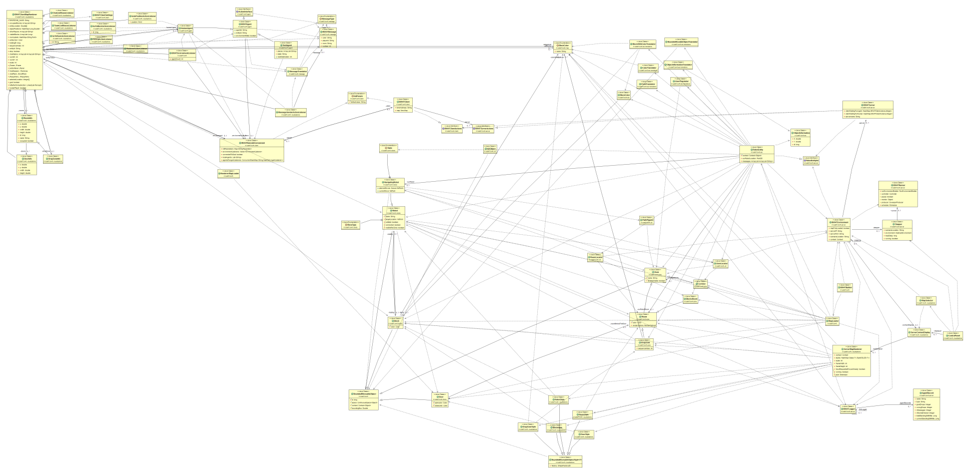
BW4T - Refactoring

June 25, 2014

1 How it was

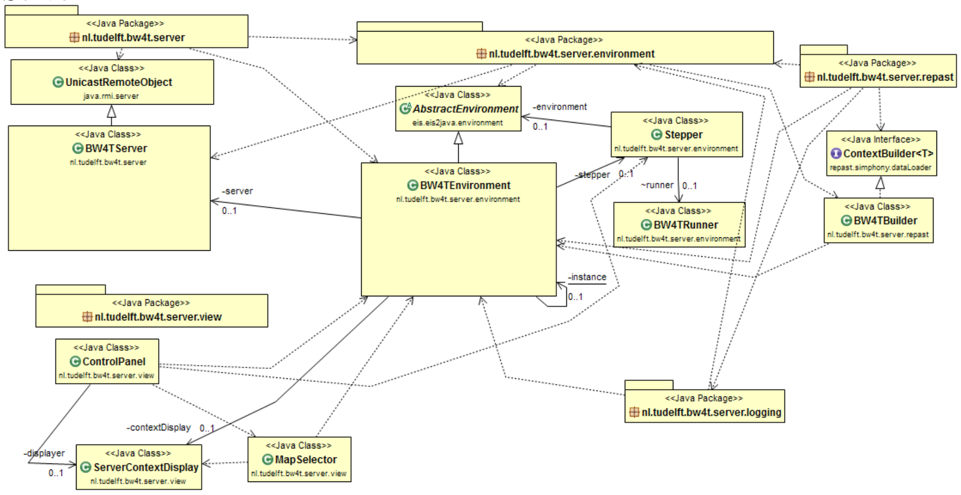


As can be seen in the UML diagram above, many classes were connected, Documentation was inadequate and no tests were written. These factors made it very hard for us to understand and debug the code base. It took quite some time to find out what a piece of code exactly did, what function a method had, what the difference between methods and classes was and what the connection between pieces of code was. After some weeks of exploring the code base and the UML diagrams belonging to the code base, we found out that there was no clear structure at all. There was a client and a server, but all communication went through the BW4TEnvironment class. So the first thing we did, was deciding how to split up the code, such that we got a clear Client-Server Architecture.

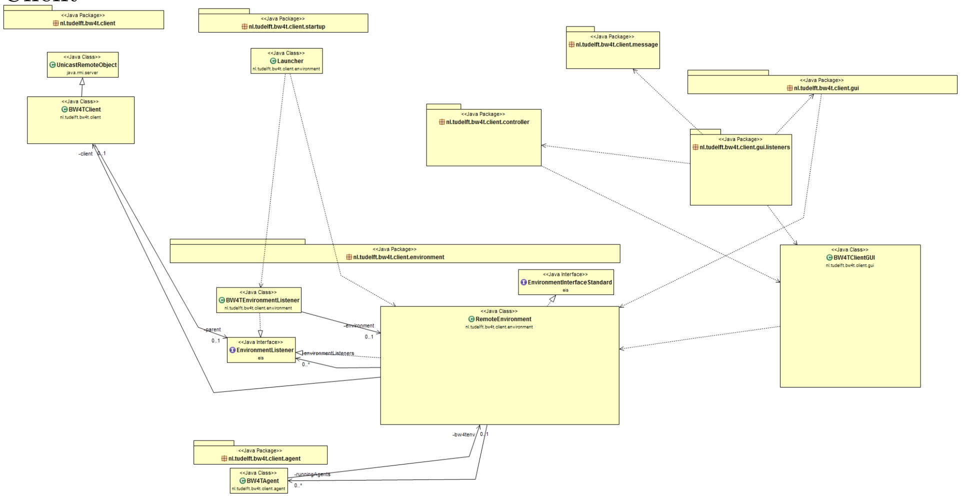
Take a look at the attachments as well for some examples of the old code. In the old system, Repast was used. Problem was that many functions in Repast were custom written as well and the entire Repast package needed to be installed. This caused the system to grow very large (500 MB for Repast only).

2 What we achieved

Server



Client



The customer wanted a clear structure to be added to the code base. Besides that, the code had to be rewritten so that it would be easier to adjust and maintain the code. Adding new features should than automatically become easier as well.

We decided to put classes which the Client needs and the Server does not need into the Client part. This does not cause any errors in the server, and anything needed by the client is present. Afterwards, we did the same for the Server. All classes needed by the Server and not by the Client are put in the Server part. The classes that are needed by both, we put in the Core part. The Core can be seen as a library, it can be used by both the Server and the Client when needed. For the Client and the Server, simplified diagrams are shown above. We chose to use Maven as an outline for the project, because Maven has a perfect way of handling those parts and the dependencies between them. Other dependencies such as EIS and Repast can be put in the resources folder and Maven takes care of the rest.

After everything was set up, we took a deeper look at the code. Trying to figure out what could be replaced by Repast was a big task that, for now, unfortunately led to nothing. Repast is a big bunch of interfaces which can guide your simulation of agents. Unfortunately, there were some problems with the connection to GOAL. GOAL can be paused and resumed whenever the user wants to, but Repast will continue to run, which will lead to false simulations and results. However because we only had 5 weeks to refactor the entire code base, we decided to leave Repast as it is for now. The chance of breaking the project was too big, as we had to deliver a fully working product in only 5 weeks.

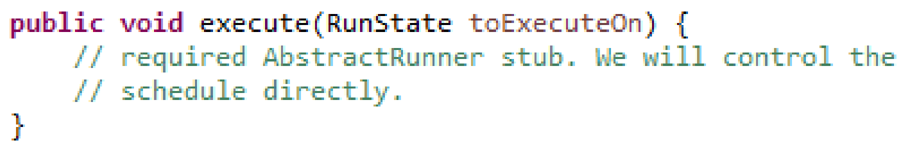
But we did manage to get the Repast functions more efficient. In combination with Maven, we pulled Repast apart and only added the dependencies that are really needed. Now the Repast functionality used is only 8 MB. Besides that, we made sure the newest version of Repast is used. So we started looking at what could be refactored. We created our own UML class diagram, and got really shocked about all connections and tried to figure out what the god classes are and how we could make them smaller. We immediately saw that BW4TRemoteEnvironment was the hugest class in the project, so that had to be refactored. BW4TClientMapRenderer was the second god class we handled. We tried to pick pieces of code that belong together and diffused it to a different class.

Then we found the custom BW4TLogger, a logger implemented to print anything the bot does. We thought that it would be very practical to export the logging to Log4j (an apache library). Now two kinds of logs are made, one kind that prints all the bot's actions to the log file and the other prints the debug issues in the code to the console. Now it can easily be adapted, if anything new from a bot is wanting to be logged, you just log it to BOTLOG level and it appears in the log file.

Attachments

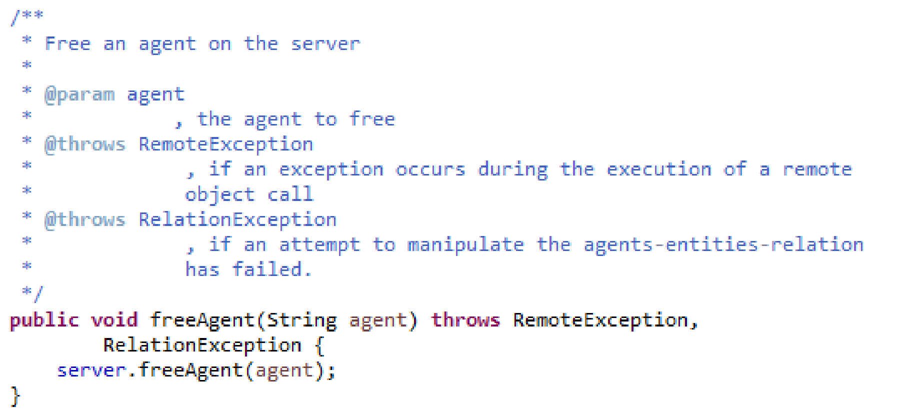
This section contains some examples of the old code base that we found. It were examples of what we wanted to adjust and make sure it never happens again.

Empty methods.

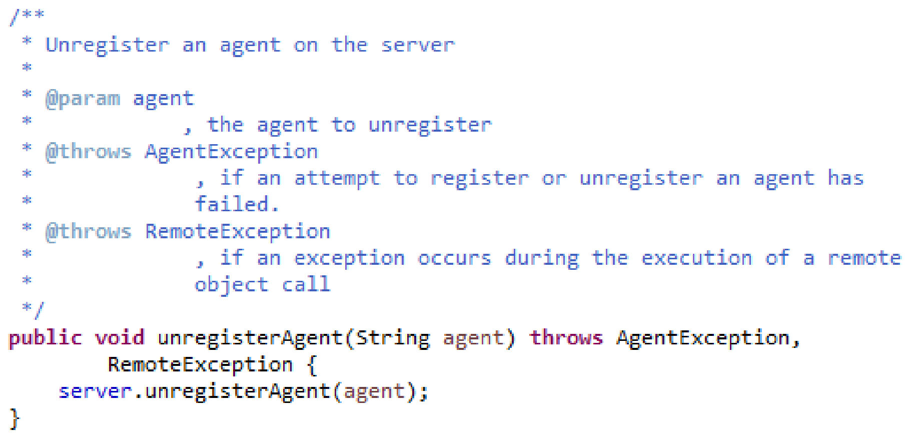


```java
public void execute(RunState toExecuteOn) {
    // required AbstractRunner stub. We will control the
    // schedule directly.
}
```

Two methods that look alike, but the difference is nowhere to be found .

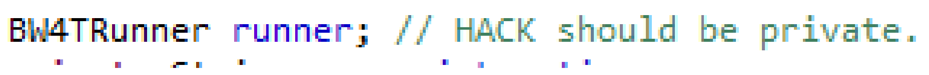


```java
/**
 * Free an agent on the server
 *
 * @param agent
 *            , the agent to free
 * @throws RemoteException
 *             , if an exception occurs during the execution of a remote
 *             object call
 * @throws RelationException
 *             , if an attempt to manipulate the agents-entities-relation
 *             has failed.
 */
public void freeAgent(String agent) throws RemoteException,
        RelationException {
    server.freeAgent(agent);
}
```



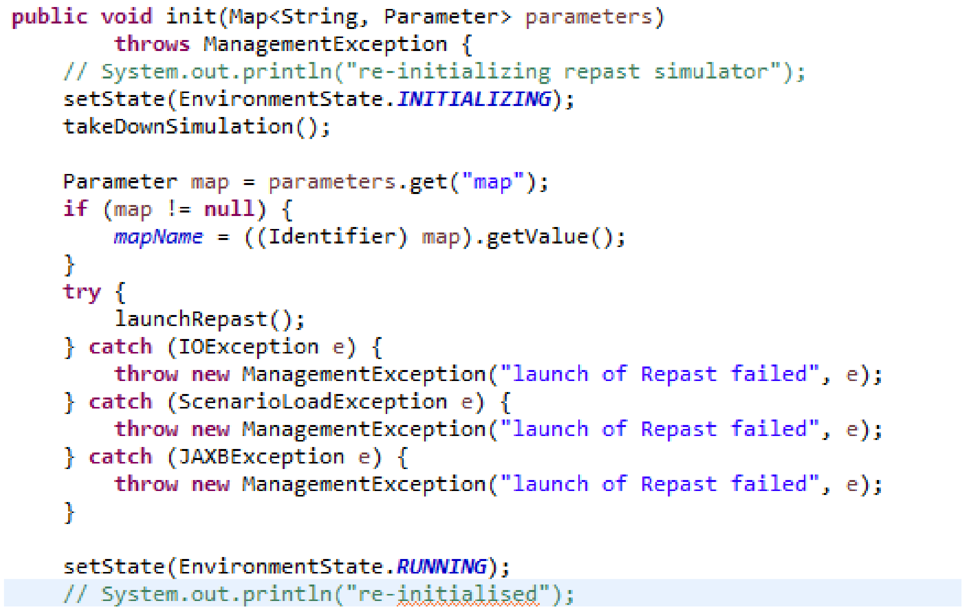
```java
/**
 * Unregister an agent on the server
 *
 * @param agent
 *            , the agent to unregister
 * @throws AgentException
 *             , if an attempt to register or unregister an agent has
 *             failed.
 * @throws RemoteException
 *             , if an exception occurs during the execution of a remote
 *             object call
 */
public void unregisterAgent(String agent) throws AgentException,
        RemoteException {
    server.unregisterAgent(agent);
}
```

Hacks



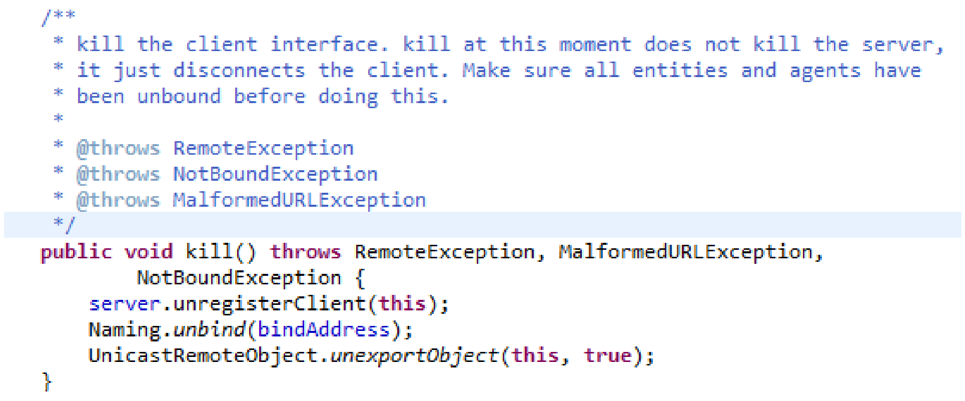
```java
BW4TRunner runner; // HACK should be private.
```

Commented code.



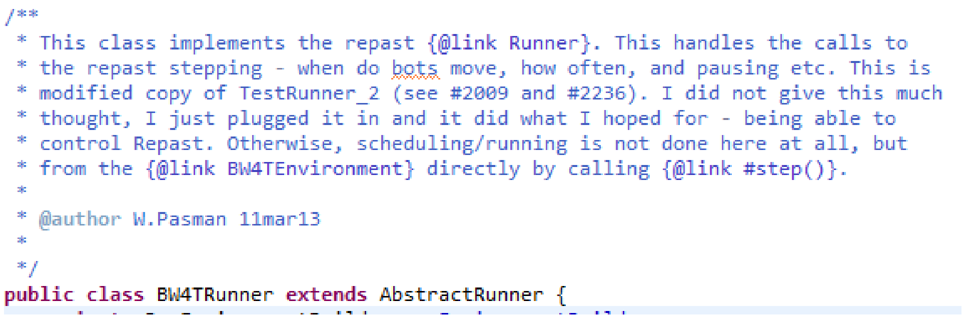
```java
public void init(Map<String, Parameter> parameters)
        throws ManagementException {
    // System.out.println("re-initializing repast simulator");
    setState(EnvironmentState.INITIALIZING);
    takeDownSimulation();

    Parameter map = parameters.get("map");
    if (map != null) {
        mapName = ((Identifier) map).getValue();
    }
    try {
        launchRepast();
    } catch (IOException e) {
        throw new ManagementException("launch of Repast failed", e);
    } catch (ScenarioLoadException e) {
        throw new ManagementException("launch of Repast failed", e);
    } catch (JAXBException e) {
        throw new ManagementException("launch of Repast failed", e);
    }

    setState(EnvironmentState.RUNNING);
    // System.out.println("re-initialised");
```

No checks whether code satisfies pre condition.



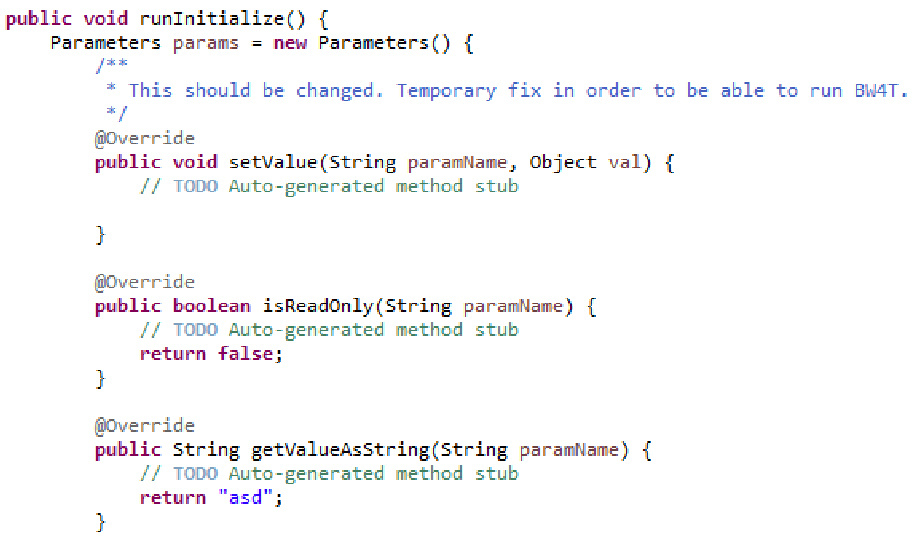
```java
/**
 * kill the client interface. kill at this moment does not kill the server,
 * it just disconnects the client. Make sure all entities and agents have
 * been unbound before doing this.
 *
 * @throws RemoteException
 * @throws NotBoundException
 * @throws MalformedURLException
 */
public void kill() throws RemoteException, MalformedURLException,
        NotBoundException {
    server.unregisterClient(this);
    Naming.unbind(bindAddress);
    UnicastRemoteObject.unexportObject(this, true);
}
```

Nothing tested, everything is assumed to work.



```java
/**
 * This class implements the repast {@link Runner}. This handles the calls to
 * the repast stepping - when do bots move, how often, and pausing etc. This is
 * modified copy of TestRunner_2 (see #2009 and #2236). I did not give this much
 * thought, I just plugged it in and it did what I hoped for - being able to
 * control Repast. Otherwise, scheduling/running is not done here at all, but
 * from the {@link BW4TEnvironment} directly by calling {@link #step()}.
 *
 * @author W.Pasman 11mar13
 *
 */
public class BW4TRunner extends AbstractRunner {
```

TODO’s and should-be-fixed’s.



```
public void runInitialize() {
    Parameters params = new Parameters() {
        /**
         * This should be changed. Temporary fix in order to be able to run BW4T.
         */
        @Override
        public void setValue(String paramName, Object val) {
            // TODO Auto-generated method stub

        }

        @Override
        public boolean isReadOnly(String paramName) {
            // TODO Auto-generated method stub
            return false;
        }

        @Override
        public String getValueAsString(String paramName) {
            // TODO Auto-generated method stub
            return "asd";
        }
```