The Zeus Agent Building Toolkit

My First Zeus Application

Jamie Stark, jamie.stark@bt.com

Intelligent System Lab, BTexacT

Release 1A, 2/3/01

Contents

Document History

First edition released 02/03/2001

1 INTRODUCTION

This document provides a tutorial for building the simplest "hello world" program using Zeus. It is aimed at the Zeus beginner, but does is not a stand alone document. In particular the reader should familiarise themselves with the following:

- Agent and other AI concepts used in ZEUS.
- Installing Zeus, running the agent builder and the examples.
- The Java programming language.
- The Zeus runtime and realisation guides.

This document is not meant to replace the existing Zeus documentation, but to provide a simple example to give the new user a flavour of how to implement a very basic agent system using Zeus.

Application Requirements

The requirements for this application are simply that the user requests a string to be outputted by pressing a button on a GUI.

Application Analysis

There will be two agents:

- InfoSupplier – supplies a string.
- Displayer – displays a string on the DOS window when requested to by the user.

And the following facts:

- **output** which has a string containing the text to be displayed
- **displayed** which has boolean flag to denote if the string has been displayed

APPLICATION DESIGN

Application design is done using Zeus's agent builder tool. Not every step is shown in detail as most of the basics are covered in the Zeus realisation guide.

Defining the ontology

The ontology is defined using the ontology editor. Define two facts:

- **output** which contains a info attribute of type string.
- **displayed** which contains a boolean flag with default value of false.

Defining the agents

The agents are first named in the agent builder. Use the two names described above in the analysis section. Once these are named go onto define each agent in turn: Remember that spaces in filenames are not handled well in the Java environment.

InfoSupplier

Under the agent definition tab give the InfoSupplier agent the ability to do the primitive task SupplyInfo. Going to the next design stage, under the agent organisation tab do nothing. Note that by leaving the agent organisation blank a Zeus facilitator will be required. Under the agent co-ordination tab the supply agent should be able to SupplyInfo without any help, so it only needs to be a respondent in the contract-net protocol. Accept the default strategy presented by Zeus.

Displayer

Under the agent definition tab give the Displayer agent the DisplayInfo primitive task to do. Again, under the agent organisation tab do nothing. Under the agent co-ordination tab choose the initiator role from the contract-net protocol since the Displayer will want some help in getting the preconditions for the DisplayInfo task. Again accept the defaults strategies presented.

Defining the tasks

During the agent definition two task were named. Using the task editor now define these tasks.

SupplyInfo

This task creates an output fact from thin air (as if by magic!). This is done by specifying that the task has an output fact as an effect. The info attribute should be set to the string you want to see outputted, e.g. "hello world". There are no task constraints.

DisplayInfo

This task actually displays the output. This functionality is added during implementation. During design we only need to specify the preconditions and effects of the task. The precondition for this task is an output fact. The only effect is that a displayed fact is created and with the attribute flag set to true.

IMPLEMENTATION

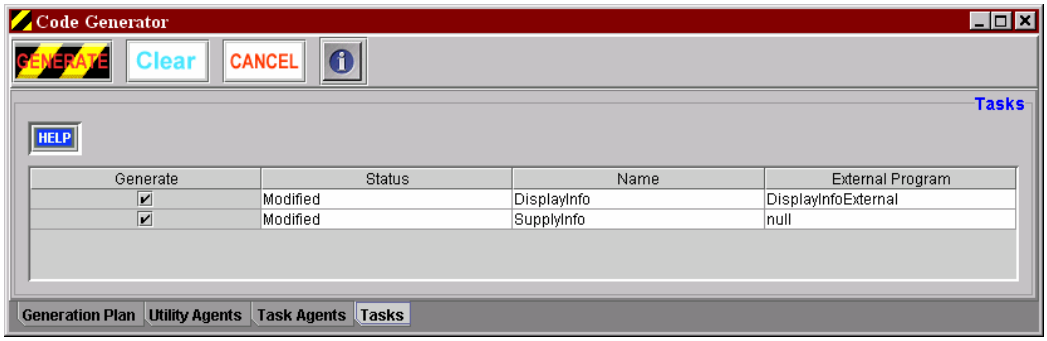
The implementation involves two steps. The first is to automatically generate the code stubs for the agents, tasks and start-up scripts from the agent builder tool using the agent generator. The second step involves writing Java code for a task external and an external GUI.

Generating the code

Using the agent generator (under the project options) we will generate the code for the agents and tasks.

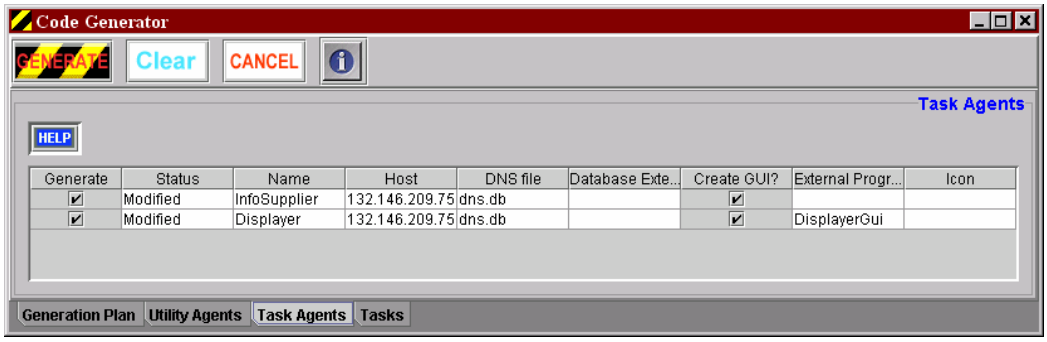
Tasks Tab

Working backwards, starting with the tasks tab specify an external program for the DisplayInfo task as a DisplayInfoExternal. We leave the other task to have a null external.



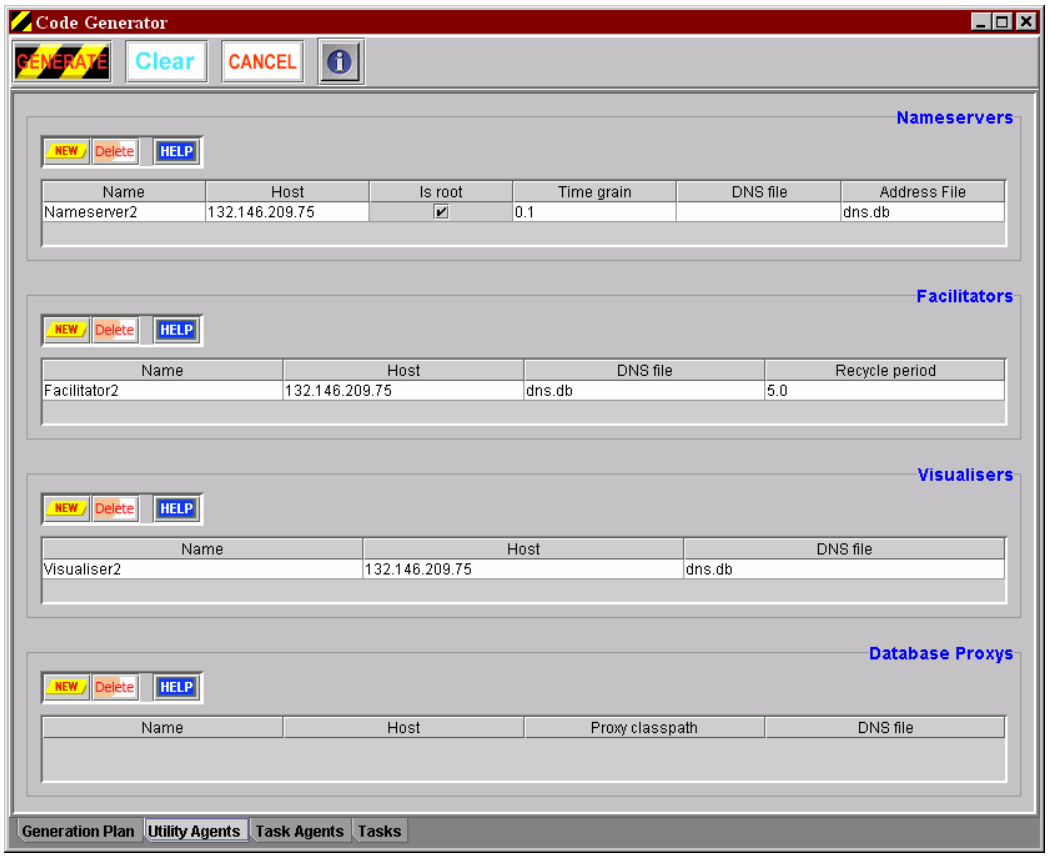
Task Agents Tab

Next choosing to create a GUI for both agents will ensure that a visualisation GUI appears for each agent helping to visualise the agents' internals. For the Displayer also name an external program `DisplayerGui`. Make sure the host is the machine you will be running the agents on.



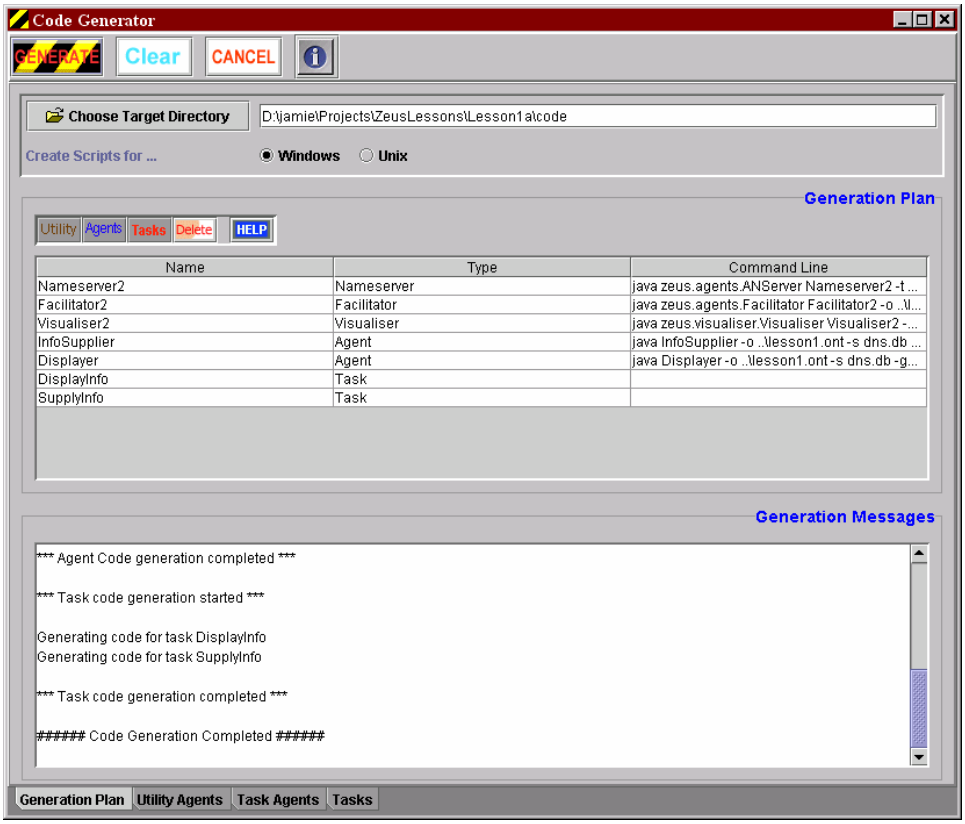
Utility Agents Tab

Accept everything here, making sure the nameserver is root and that the host is the machine you will be running the agents on.

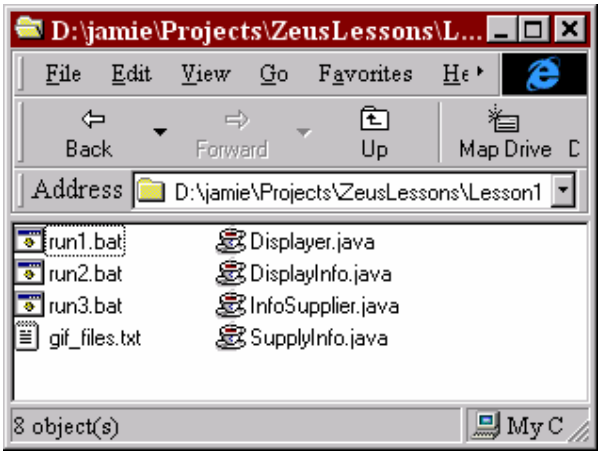


Generation Plan Tab

Choose the target directory for the code to be generated in and which platform you are using. A known bug is that if you have already specified the target directory then Zeus may generate into the old directory. To rectify this you must save the project, shut down Zeus, restart and reload the project. Selecting the directory should now work, but you will need to check that Zeus remembered all the external programs you have just defined. Another known related bug is that the file browsing capability does not seem to work in the way expected.



Now hit the generate button. This should create the following files:



The batch files are scripts that start up the various agents and services

- run1.bat starts up the nameserver.
- run2.bat starts up all the agents.
- run3 starts up the facilitators and visualiser.

The java files define the definitions of the agents and the tasks. Now we must implement the external programs we named in the agent generator above.

Implementing the agent external

The purpose of the `DisplayerGui` external is to provide a button for the user to push. When the button is pressed a new goal should then be asserted into the Displayer agent. To work with Zeus this

class must implement the `ZeusExternal` interface. This means that it must implement a method called `exec`.
The code is included below.

```
// I am not sure if all the zeus imports are required for this example,
// but they are likely to be needed for more complex examples
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import zeus.agents.*;
import zeus.util.*;
import zeus.concepts.*;
import zeus.gui.fields.*;
import zeus.actors.*;
import zeus.actors.event.*;
import zeus.generator.util.*;

/**
* DisplayerGui enables a user to press a button that asserts a
* displayed goal into the agents context database.
* Note this is research quality software.
* @author Jamie Stark Jan 2001
*/
public class DisplayerGui extends Frame implements ZeusExternal{

    /** The agent context enables the external to get at the attributes
        of an agent */
    private AgentContext agent;
    /** The Button to push */
    private Button button;

    /** Constructor sets up the frames and add an ActionListener to
        the button*/
       public DisplayerGui(){
        button = new Button("Assert Goal");
        add(button);
         setSize(290,155);
         setVisible(true);
         button.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent event){
                       Object object = event.getSource();
                       if (object ==button)
                              assertGoal();
            }
               });
       }

   /**
   * This is called as the agent is created. Here all that happens is
   * that AgentContext is stored
   */
    public void exec(AgentContext agentContext){
        agent = agentContext;
    }

    /** This method does all the work. It creates a new displayed fact,
    * sets the value of the flag attribute to true, creates a new goal
    * using this fact and asserts it into the agents co-ordination engine.
    */
    public void assertGoal(){
      //Create a new fact using the ontology.
      Fact fact = agent.OntologyDb().getFact(Fact.FACT,"displayed");
      // Set he attribute flag to true.
      fact.setValue("flag","true");
      // Create new Goal.
      Goal g = new Goal(agent.newId("goal"), fact,(int)(agent.now())+6,
                        0.1,agent.whoami(),(double)(agent.now()+3));
      // Assert the goal in the agent's co-ordination engine.
          agent.Engine().achieve(g);
    }

}
```

Implementing the task external

The purpose of the DisplayInfoExternal class is to implement the functionality of the DisplayInfo task. This can be achieved by writing the info attribute of the output fact to System.out. All the important information that is required is held in the TaskContext object. The code below shows how to get this information out.

```
// I am not sure if all the zeus imports are required for this example,
// but they are likely to be needed for more complex examples.
import zeus.agents.*;
import zeus.util.*;
import zeus.concepts.*;
import zeus.gui.fields.*;
import zeus.actors.*;
import zeus.actors.event.*;
import zeus.generator.util.*;


// Note that our External must implement the TaskExternal Interface
public class DisplayInfoExternal implements TaskExternal{

    // To implement the TaskExternal we must implement the exec method
     public void exec(TaskContext tc){
        // The taskContext's inputArgs is an array Fact arrays.
        // There is only one input so we want the first,
        // i.e. get the first precondition which is an outputFact
        Fact[] outputFact = tc.getInputArgs()[0];
        // We then get the outputFact
        // Which is an array. Since there is only one fact it is the
        // first one
        // We print the value of the info attribute.
        System.out.println("****************************");
        System.out.println(""+outputFact[0].getValue("info"));
        System.out.println("****************************");
    }
}
```

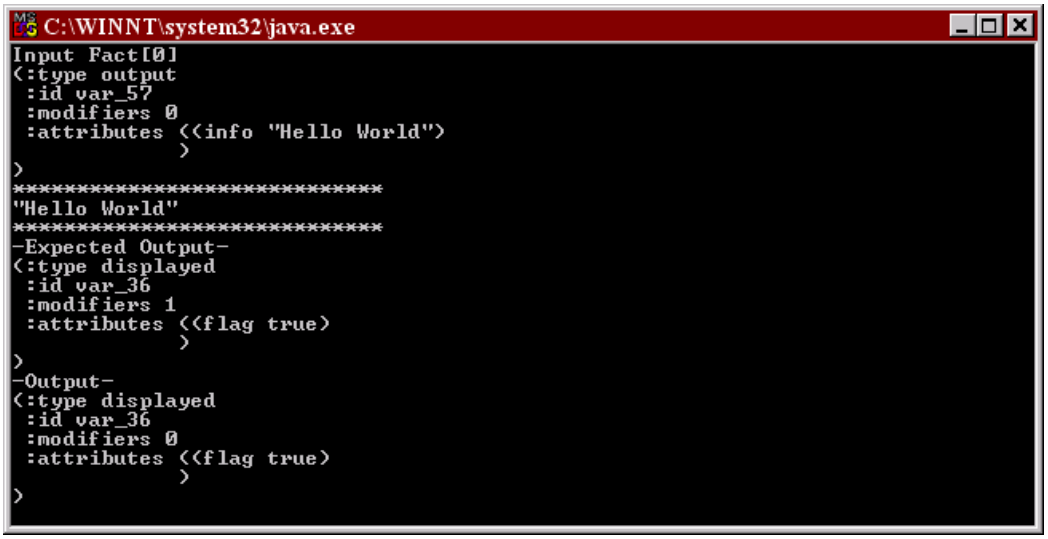
RUNNING THE SYSTEM

First all the automatically and manually created files need to compiled using javac. Now the application can be started by first running up the Zeus NameServer, then the agents and finally the Zeus facilitator and visualisation tool. These can be done using the run1, run2, and run3 batch files. Make sure these files are started in this order.

If all went well then you should have four new windows. There will be:

- A visualisation GUI for each agent. These allow you to see the various internals of the agent. Look inside the mail box and see that agents have registered with the facilitator.
- A visualisation GUI for the whole system. This can be used to view various things, including the agent interactions, which are useful for debugging.
    - To view the messages being passed around click on the Society icon. In the new window choose the centred view style. From the Online menu option choose Request messages. Hit the 'select all' button and the OK button.
    - If you click on the control icon and then on the bomb icon in the new GUI you will be able to kill all the agents and stop the system.
- The GUI from the agent external we implemented earlier. It should look something like:

Pressing this button will cause a series of message to fly around as the agents co-ordinate to solve the goal. If the agents were successful you should see something like:



as the Displayer agent executes the DisplayInfo task.

If you are having problems some useful hints are:

- Check your classpath – see the Zeus documentation.
- Do not start the java programs in the background. This running the java commands in the batch files directly in the command prompt. This will allow you to see any error message thrown by the programs.
- Do not be in too much of hurry when starting up the agents, leave a couple of heart beats between running each batch file.

WHAT NEXT?

Now you have you very first Zeus program you will probably want to experiment with it. If so here are some ideas for extensions you may want to try:

- Change the SupplyInfo to take input from an external source or the user.
- Introduce two types of info and add another SupplyInfo agent so that each agent supplies a different type of info.
- Give the user the choice about which type of information they want displayed.