Chapter 6

Implementation

6.1 Introduction

This chapter will firstly discuss the major technologies used to implement the multi agent based urban public services locating system. After that, the implementation details of each module which was identified in the design part of the previous chapter will be explained. Finally, the features of the implemented prototype will be discussed.

6.2 Major technologies used

The overall implementation of the urban public services locating system consists of module wise implementation of components that was identified in the previous chapter. Those are the implementation of terrain module, building services module, water services module, natural services module and transportation services module. These modules were implemented using NetBeans IDE 6.8 with the support of Madkit multi agent system framework 4.2.0 and protégé ontology development software 3.4.3. Each of the modules were programmed as separate classes in the Java package. The functionalities inside each of these modules will be implemented as methods inside the classes. Interactions between each of the module were also implemented as per the top level architecture diagram indicated in the previous chapter. Sample screenshots of the Implemented system are available in Appendix B.

6.2.1 Madkit

MadKit is a modular and scalable multi agent platform written in Java and built upon the AGR (Agent/Group/Role) organizational model: agents are situated in groups and play roles. It allows high heterogeneity in agent architectures and communication languages, and various customizations.

MadKit communication is based on a peer to peer mechanism, and allows developers to quickly develop distributed applications using multi agent principles.

Agents in MadKit may be programmed in Java, Scheme (Kawa), Jess (rule based engine) or BeanShell. Other script language may be easily added.

MadKit comes with a full set of facilities and agents for launching, displaying, developing and monitoring agents and organizations.

MadKit is free software based on the GPL/LGPL license.

6.2.2 AGR Model

In the AGR model, an organization is viewed as a framework for activity and interaction through the definition of groups, roles and their relationships. By avoiding an agent-oriented viewpoint, an organization is regarded as a structural relationship between a collection of agents. Thus, an organization can be described solely on the basis of its structure, i.e. by the way groups and roles are arranged to form a whole, without being concerned with the way agents actually behave, and multi-agent systems will be analyzed from the “outside”, as a set of interaction modes. Thus, the specific architecture of agents is purposely not addressed.

6.2.3 Protégé

Protégé is a free, open-source platform that provides a growing user community with a suite of tools to construct domain models and knowledge-based applications with ontologies. At its core, Protégé implements a rich set of knowledge-modeling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats. Protégé can be customized to provide domain-friendly support for creating knowledge models and entering data. Further, Protégé can be extended by way of a plug-in architecture and a Java-based Application Programming Interface (API) for building knowledge-based tools and applications.

An ontology describes the concepts and relationships that are important in a particular domain, providing a vocabulary for that domain as well as a computerized specification of the meaning of terms used in the vocabulary. Ontologies range from taxonomies and classifications, database schemas, to fully axiomatized theories. In recent years, ontologies have been adopted in many business and scientific communities as a way to share, reuse and process domain knowledge. Ontologies are now central to many applications such as scientific knowledge portals, information management and integration systems, electronic commerce, and semantic web services.

6.3 Implementation of the Graphical User Interface (GUI)

During the implementation stage, the GUI was implemented with three separate panels inside a single Java frame. The first panel includes a toolbox with image icons, while the second panel creates the virtual city environment and the third creates a message panel.

6.3.1 Toolbox Panel

The first panel includes a toolbox with image icons to represent physical objects in the virtual city environment such as buildings, water resources, natural resources and transportation resources. Each of these images represents public service agents of the system. To place a public service in the virtual city environment, these images are clicked and selected and then a position in the virtual city environment panel is clicked. When this is done, the software pertaining to each public service is invoked and the corresponding agent is created.

The toolbox image icons in the first panel were implemented as JButtons. Each JButton consists of an image and its caption to identify them easily by the user. The image is made to represent the public service and identify it easily by the user of the system. Functionalities of these image icons were implemented using the ActionListener and MouseListener functionalities of the Java AWT package. When a user presses on one of the JButtons in the toolbox, the program will call the MousePressed() event and the button will behave as a draggable component. When the user has released the dragged button in the second panel, the program will call the MouseReleased() event and create the resource while calling its input-GUI. The input-GUI is used to input type and environment of the public service. This input-GUI is used in the case of all public services except the Roadline service.

6.3.2 Virtual City Environment Panel

The implementation of the second panel is to facilitate the user to create the virtual city environment. When the system is invoked, the terrain data agent reads the terrain data file and paints the height information of each and every pixel in the Virtual City Environment

Panel. Each pixel is having dimensions of 20 m X 20 m. The terrain data file consists of height information of 250,000 pixels. The data in the terrain data file is arranged as 500 rows. In each row the height data are arranged with white space separator. The minimum height is 0m while the maximum height is 1650m. When the height information is painted in the Virtual Environment Panel, a separate and distinct colour is assigned to each and every height. The height data can take only positive integer values.

The colours assigned for heights are arranged according to three colour bands. For each integer from 0m to 499m height values different shades of green are assigned. For each integer from 500m to 999m height values different shades of Yellow are assigned. For each integer from 1000m to 1499m height values different shades of brown are assigned. For higher heights, single colour of magenta shade is flatly assigned.

The zero point of the Virtual City Environment Panel is situated in the top left hand corner. Thus the x and y coordinates are zero at that point. The x- coordinates progressively increase from left to right side while the y-coordinates progressively increase from top to bottom. Thus the extreme point of the terrain with coordinates (500, 500) is situated at far right and bottom-most corner.

Whenever the Road sign in the Toolbox panel is clicked and dropped, a roadline agent is created. To position a road, in the Virtual City Environment, the mouse will have to be traced between the two extremes of the desired road without releasing the mouse button. Then the methods pertaining to the Roadline class will be invoked and the road is drawn on the Virtual City Environment with a red clour line of thickness 3.

Whenever an icon other than the Road sign in the Toolbox panel is clicked, dragged and dropped on the Virtual City Environment, an input-GUI box will open with drop down menus for the type of the public service and the environment of the public service. The parameters depending on the public service may be rural or urban of environment and commercial, national, international, rural, central etc of type. After selections are done at the input-GUI box, the classes of the corresponding agent are invoked to draw the symbol of the public service on the Virtual City Environment.

Depending on the rules assigned to public service, the values of the parameters assigned to it, and the availability of other already existing public services within the Virtual City Environment, certain interactions will take place. As a result the new public service is moved by the system to a new location, it is allowed to remain in the user-assigned position or the movement is abandoned having failed to find a suitable alternative location. In such an extreme situation, the user is advised to relocate the public service manually.

6.3.3 Message Panel

Just after the initialization of the system along with the creation of the Virtual City Environment Panel, a Message Box Panel is created in the left hand side. The messages pertaining to the creation or killing of agents and all the communication between the agents and the instruction messages that are meant to the user of the system are displayed on this Message Panel. The user will be able to monitor the functioning of the system and especially the dialogues that take place between the agents using this message panel. The messages will be of great importance to users whenever a user is required to manually intervene especially in the case of a requirement to manually place a public service resource on the Virtual City Environment sequel to killing of an agent.

6.4 Implementation of agents

The public service agents who are belonged to building services module, water services module, natural services module and transportation services module were implemented inside the Madkit agent framework and it contains set of classes to represent agent behaviours. All the agents inherit the primary attributes and roles from the TestAbstractAgent class, which is the abstract class of the program. This class was implemented by extending the attributes and methods of the madkit in-built agent class. The TestAbstractAgent class contains the functionalities to create a group of agents, assign them a role, check whether their performing the task and to destroy them after completing their execution.

6.4.1 Terrain Agent

When, the system is loaded, terrain data is fed into the system reads the terrain data file. This data consists of terrain height against the location coordinates. As indicated earlier the data consist of height details of 250,000 pixel points. These data correspond to average height of a pixel of dimensions 20 m x 20 m. As mentioned earlier these data are read by the Terrain agent and coloured rectangles of corresponding colours are painted on the

Virtual City Environment panel. At the same time the Message panel is indicated with the message that the Terrain Agent is created.

Once a public service agent is created on the Virtual City Environment Panel, the location data is known to them. They will be heavily dependent on the terrain agent to find the height of their position whenever it is required. This height data is very important for the Public Service Agents. This sample code segment is to show the terrain agent's response for a request by a public service agent,

```
while (live) {
        Message msg = waitNextMessage();
        NetMessage n = (NetMessage) msg;
        if (n.getMsg().equals("sendHeight")) {
         sendMessage((AgentAddress) n.getSender(), new NetMessage("height " +
        getHeight1()));
        }
```

to calculate their rate of decay of influence that depend on their height. Say, a hospital of a certain type and environment located on a hill will have wider influence on the other public service agents when compared to the same type and terrain hospital on a flat ground. Some heights are not suitable or prohibitive for certain public service agents.

In the special case of Roadline agent, dependence on the terrain agent is very heavy. For finding the gradient of a road segment a Roadline agent will regularly query the terrain agent.

6.4.2 Roadline agent

Whenever the icon corresponding to a road is clicked on the toolbox panel and the starting and end points of the intended road is drawn on the Virtual City Environment panel without releasing the mouse, the Roadline Agent class will be invoked and its methods are executed. Knowing the coordinates of the start and end points, the Roadline agent will draw a line connecting the two points. Then it will find out the coordinates of the first segment at 20 m distance along that line from the starting point. Subsequently, the Roadline agent will query the heights of the start and end points of the segment and use those data to calculate the gradient of the segment. Following sample code shows how the gradient is calculated by Roadline agent,

```
public void calculateGradient(int h1, int h2) {
        double gradient;
        double lastDistance;

        gradient = Math.abs((h2 / 200 - h1 / 200) / Math.sqrt((rFromX - rToX) * (rFromX -
        rToX) + (rFromY - rToY) * (rFromY - rToY)));
        mg = " ROADLINE AGENT : I have calculated the gradient of the road
        segment."+" Its value from (" + rFromX + "," + rFromY + ") to (" + rToX + "," +
        rToY + ") is " + gradient;
        MapFrame.textArea.append(mg + "\n");
                 if (gradient <= 0.125) {
                double stAngle = calAlpha(rFromX, rFromY, rToX, rToY);
                mg = " ROADLINE AGENT : As the Gradient is less than 1/8, I am
                proceeding along the segment...";
                MapFrame.textArea.append(mg + "\n");
                rFromX = rToX;
                rFromY = rToY;
                }
        }
```

while, the gradient is smaller than 0.125, either positive or negative, the Roadline agent will mark the segment as OK for future implementation. If the gradient surpasses the threshold value, the Roadline agent decides to shift the segment by 0.005 degrees either in the increase of the angle and then in the direction of decrease of angle. This change of directions occurs after shifting in a certain direction for 5 degrees corresponding to 1000 shifts.

The angles are measured from negative Y-axis counter-clock-wise. The angle will be 0 degrees to 180 degrees in that direction and will be negative 0 to negative -180 degrees in the clock-wise direction.

The shifting of the road segment through incrementally increased small angles will be done until the gradient condition is met. After each shifting, the coordinates of new end point of the segment is determined and its height is queried. If a suitable angled segment could not be found even after extreme positive and extreme negative shifting of angles, the user is notified about the inability of drawing the road. The roadline agent is killed. The user will have to select a fresh two new points.

In the event of success of finding a suitably angle shifted road segment with gradient condition satisfying, that segment is ear-marked by the Roadline agent as OK for future drawing.

After successful drawing of a road segment, the portion of the road in between the end point of the completed the segment and the original end point of the road is executed repeating the above mentioned procedure.

6.4.3 Public Service Agents other than Roadline agents

Whenever a public service agent other than a roadline agent is created sequel to drag and drop of the corresponding symbol from the Toolbox panel to the Virtual City Environment Panel an agent of the corresponding public service is created. As indicated earlier, the agent requests from the user its type and environment details.

This sample code illustrates how a newly created public service agent informs other agents in the city about its arrival,

```
        public void sendMsg(){
            if (MapFrame.locations.size() > 1) {
                sendMessage(Location.COMUNITY, Location.GROUP, Location.ROLE,
                new NetMessage("NewBuildingCreated"));
                 }
                if (MapFrame.roads.size() > 0) {
                sendMessage(RoadLineAgent.COMUNITY, RoadLineAgent.GROUP,
                RoadLineAgent.ROLE, new NetMessage("NewBuildingCreated"));
        }
   }
```

queries and finds its height and keeps it in case any other agent queries it. The agent is assigned a primary value or strength depending on its variety, viz. Bank, School, Hospital etc. Having found out its type and environment, the agent computes the new value of its value or strength. For example a rural Hospital may have a smaller value than a National Hospital.

This value or strength is for unaccompanied situation, when only this agent is present in the Virtual City Environment. Depending on the other agents present around him and their proximities, the Value or Strength of the agent will change. The agent calculates this new value using the equations assigned to him. He keeps this value to be used in case any other agent queries for it.

The agent will also have a rate of decay of influence. This will also have a base value depending on the variety of the agent. Its new value will depend on type, environment and of course the height of its location. The agent will calculate and keep this new rate of decay of influence in case another user will query for it.

The agent will also have a tolerable influence level. Here there will be several values depending on from which he tolerates the influence. Tolerable influence for a School from a Hospital will be different from a Tolerable influence from a Super-Market. Here again the assigned values are base values which are adjusted by the agent depending on its type and environment. For example a rural school will tolerate more from a super market than a national school. This new values of tolerable influence from other agents will be calculated and kept by the agent for its own use. These values will not be queried by other agents.

The newly created agent then queries the value, position and rate of decay of influence of each of the already existing agents. This too is done according to a certain rule. Not all the other agents influence a particular agent. A particular agent does not influence all the agents. Depending on these two sets of information, the new agent queries the above information from certain existing agents.

Then the agent calculates the influence of that agent at a distance of 1 km from the agent in the radial direction towards the existing agent using the equation,

$$Influence_{secondAgent} = Value_{firstAgent} - (RateofDecay_{firstAgent} * (distance - 1))$$

and if the calculated influence is smaller that the tolerable influence from the category of existing agents, that existing agent is earmarked as OK.

This is repeated for all existing agents. If any of the existing agents prove surpassing the tolerable value of influence, then the whole exercise is aborted. The system notifies that the position of the new agent is not suitable. It will earmark the area the new agent can shift and automatically shift is to a new position.

Then the whole procedure is repeated. If even that prove futile, another position is sought. If all fails, the user will be notified to downgrade the type and environment and relocate the public service manually, which will again commence the whole procedure.

If the new agent finds that the influence due to all the other agents are within tolerable limits, then the second phase will commence.

During the second phase, each existing agent will query data from the new agent and find whether the influence is tolerable. If the test fails the system will go for abort mode or if the test passes, the new user will be allowed to stay in the location.

All the agent details will be stored in the ontology. All the agents will be destroyed automatically from the system when they have completed their individual and group roles and updated the ontology.

6.5 Implementation of the Ontology

The public service ontology was implemented using the protégé software 3.4.3 version to store the initial knowledge about all the agents. Public service ontology contains classes to store knowledge on building service agents, water services agents, natural services agents and transportation services agents. The protégé-owl api and jena plugin for protégé was used to connect the java program with the ontology by implementing a bridge between the ontology model that is created inside the program and the .owl file generated from protégé software. The ontology was created by inserting separate classes for each agent and introducing separate data type properties for each of the class. Once a public service is located inside the virtual city environment an instance of the agent is created inside the ontology referring the respective class. This instance will assign the data type properties of the respective class and will transfer the values for those properties from the program.

6.6 Features of the solution

The prototype was implemented to cover most of the features identified in the chapter 4. The following part of the chapter contains the implementation details of each feature.

Feature 1 – Locating public services in a given city environment can be implemented with the use of negotiation feature of multi agent technology. Once the virtual environment is created by the user and a new public services agent is created in the city environment, the agents that are already in the environment will be initialized and negotiate accordingly to find a suitable location for the new public service agent.

Feature 1 was implemented by using the building services module, natural services module, water services module and the transportation services module. Whenever the user selects an icon of a public service and clicks a point in the map, an agent of the corresponding public service is created. Certain details such as the type and environment are fed through a dialog box. The terrain agent locates and identifies the location coordinates of the newly created public service agent. The other existing public service agents having noticed the creation of the new agent commence communication with him. The location, the value, the rate of decay of influence will be queried. Each existing agent will find out whether the influence of the new agent will surpass the tolerable level of influence. If any of the existing agents find that their tolerable influence is violated, then the system will message the newly created agent to move from the position. The system will suggest and move the newly created agent to a new position. And after that fresh communication will take place.

Feature 2 - The beginners of urban planning can analysis the interactions between the public service agents while trying to locate the new public services inside the city environment.

Feature 2 was implemented by allowing the public service agents to communicate and negotiate while finding for a suitable location for the newly created public service. The conversations between the two public service agents are displayed in the message space.

Feature 3 - Details of the finalized locations of the public service agents will be stored in the ontology for future reference.

Feature 3 was implemented by creating a Jena ontology model inside the program referring the .owl file generated from protégé software. The ontology model contains the hierarchical structure of classes which refer agents of the program. The public services agents are created with the use of the initial knowledge stored in the ontology and will update the ontology when the agents have executed their tasks.

Feature 4 - Agents consider not only one public service but dependency between many public services. Those are transportation services, buildings services, water services and natural services.

Feature 4 was implemented in a situation where the user creates few public services that belong to several categories such as buildings, water resources and natural resources. Then those public service agents will start to negotiate while calculating the influence between each other.

Feature 5 - Further, memory consumption and hardware capacity during the processing is very limited because agents are created when required and will be destroyed when their task is complete.

Feature 5 was implemented by setting a life time for each agent that are activated in the program. When the life time ends, agents will be killed. When a request comes, a new agent will be initiated.

6.7 Summary

At the beginning of this chapter, the major technologies used to implement the prototype were identified. The implementation details of each module were explained secondly, and thereafter, the features of the implemented prototype were discussed.