Collecting Gold
MicroJIAC Agents in MULTI-AGENT PROGRAMMING CONTEST

Erdene-Ochir Tuguldur and Marcel Patzlaff
tuguldur.erdene-ochir@dai-labor.de
marcel.patzlaff@dai-labor.de

DAI-Labor, Technische Universität Berlin, Germany

1 Introduction

The framework used here is the result of the diploma thesis "Development of a Scalable Agent Architecture for Constrained Devices" written at DAI-Labor which is due to be finished in March 2007. This contribution provides a first bigger implementation with microJIAC. The motivation to participate on the contest is to test the functionality and usability of the framework. Since the problem leaves enough space to experiment, a solution is found that fits in microJIAC's agent model and exploits most of its capabilities. All test results will be evaluated within the scope of the diploma thesis mentioned above to draw conclusions and to propose future development of the framework.

2 Software Architecture

2.1 MicroJIAC Framework

The microJIAC framework is a lightweight agent architecture targeted at devices with different capabilities. It is intended to be scaleable and useable on both resource-constrained devices (i.e. cell phones and PDAs) and desktop computers. It is implemented in the Java programming language. At the moment a full implementation for CLDC[1] devices is available, which is the most restricted J2ME[2] configuration that is supported.

The agent definition used here is adapted from [1]. It is a biologically inspired definition where agents are situated in some environment and can modify it through actuators and perceive it through sensors. Thus the framework is also split into environment and agents. The environment is the abstraction layer between the device and agents. It defines lifecycle management and communication functionalities. These functionalities include a communication channel through which the agents send their messages.

Agents are created through a combination of different elements. The predefined element types are Consumers, Producers, Rules, Services and Components.

[1] Connected Limited Device Configuration `http://java.sun.com/products/cldc/`
[2] Java 2 Platform Micro Edition `http://java.sun.com/javame/index.jsp`

Consumers and Producers are the interface between the agent and the environment. The former resemble actuators through which the agent manipulates the environment — they consume data from the agent. The latter instead resemble sensors through which the agent gains knowledge from its surroundings — they produce data for the agent. Rules specify reactive behaviour and Services define an interface to provide access to specific functionalities. Finally, Components maintain a separate thread and host time-consuming computations. All elements are strictly decoupled from each other and are thus exchangeable. Only Consumers are allowed to define handles through which other elements may communicate with them. These handles are derived from the Connection interface of the GCF[3].

In contrast to JIAC IV [2], which is used by the other contribution of our institute, microJIAC does not use ontologies, goals or an agent programming language such as JADL [3]. Furthermore, agent migration is restricted to Java configurations which support custom class loaders and reflection. It should not be left unmentioned that beside the similarity in the names both architectures are targeted at different fields of application and have different development histories. Of course it is planned to use a common communication infrastructure to enable information exchange between each others agents.

2.2 MicroJIAC Methodology

The following steps have to be taken to realise an agent:

1. Define actuators by implementing the Consumer interface.
2. Define sensors by implementing the Producer interface.
3. Reactive behaviour is realised through defining Rules.
4. To provide functionalities that are accessible from other agents, implement the Service interface.
5. For extensive computations the Component interface has to be implemented. Those elements get an exclusive thread for their execution. They are not available for some Java configurations.
6. Agents are not implemented directly but are configured with specific XML files. In these files the agent elements (i.e. the implementing classes) are listed and customised through properties. It is also possible to modularise the configuration files by using abstract agents and inheritance of them.
7. Depending on the Java configuration the packaged implementations are deployed to a running environment or environment and agents are packaged and deployed together (in the case of CLDC).

2.3 Tools

The standard build tool for microJIAC is Maven[4] which eases the dependency and project management. Maven is based on a component architecture imple-

[3] Generic Connection Framework `http://developers.sun.com/techtopics/mobility/midp/articles/genericframework/`

[4] Maven `http://maven.apache.org/`

mented in Java and is extensible through plug-ins. MicroJIAC comes with its own Maven plug-in which supports the compilation and packaging process of agents targeted at CLDC devices. This is needed to generate several classes which circumvent the absence of Java reflection. Furthermore it can reduce the size of byte code by using the ProGuard[5] obfuscator.

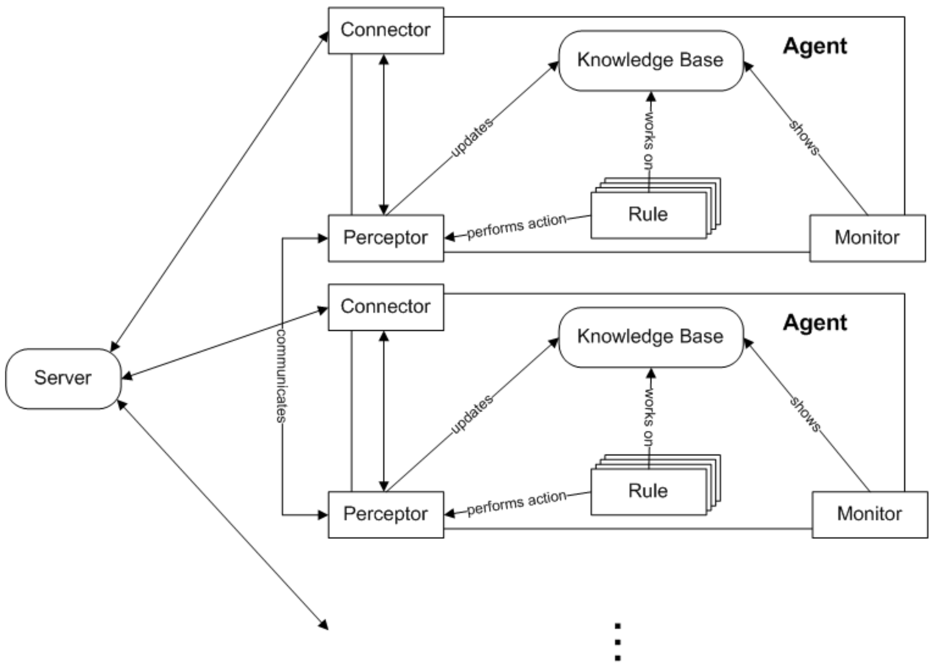


Fig. 1. Design of the Competition Agents

3 System Analysis and Design

This contribution to the contest implements a multi agent system whose agents are reactive and autonomous. These agents consist of four main agent elements (see Figure 1).

1. Connector
   It maintains the connection to the competition server. Parsing the messages received from and sending the actions back to the server are the main tasks of this element. The concrete implementation is a combination of the Producer and Consumer interfaces.

[5] ProGuard Obfuscator `http://proguard.sourceforge.net/`

2. Perceptor
   It updates the world model and fires notification events to trigger the rules. Furthermore it is responsible for the communication and coordination with the other agents. This element also implements the Producer and Consumer interfaces.
3. Rules
   These are associated to specific world model states. Depending on the state the rules create actions for the agent.
4. Monitor
   It provides a graphical user interface which displays the world model of the agent. This is used mainly for debug purposes.

3.1 Strategy

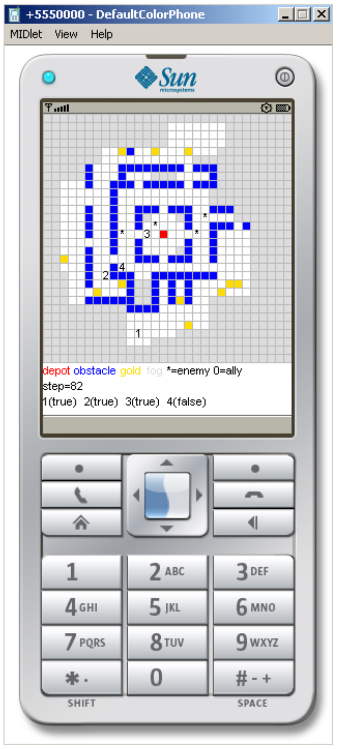
We have currently the following rules:

1. The agent stands on the depot.
   - If it carries a gold item, drop the gold item.
   - If there are no gold-containing or unknown cells nearby, use the teleport function.
2. The agent does not carry a gold item.
   - If it stands in a cell which contains a gold item, pick the gold item and compute the shortest path to the depot with A* algorithm.
   - If there are no gold-containing or unknown cells, mark all cells as unknown.
   - If there is at least one gold-containing or unknown cell which is not a destination of other agents, choose the nearest of them as a destination and compute the shortest path to it.
   - If available follow the precomputed path.
   - If an unexpected gold item appears on the way, determine the destination again and compute the shortest path to it.
   - If an unexpected obstacle appears on the path, compute the shortest path again.
3. The agent carries a gold item.
   - Follow the precomputed path to the depot.
   - If an unexpected obstacle appears on the path, compute the shortest path to the depot again.

3.2 Communication and Coordination

The agents use message based communication. The underlying protocol — Stomp[6] — distinguishes between queue messages and topic messages. Topic messages are used to distribute changes of the agent's world model to the team. So every agent should have the same view on the surroundings. Coordination is reached as every agent also communicates its destinations.

[6] Stomp Protocol Specification `http://stomp.codehaus.org/Protocol`



```
depot obstacle gold fog *=enemy 0=ally
step=82
1(true) 2(true) 3(true) 4(false)
```

Fig. 2. An agent runs on SUN Wireless Toolkit

4 Conclusion

A framework targeted at small devices might not be the first class choice to realise the given scenario with. But is a good example to see which capabilities those devices offer and how they can be exploited with the proposed system.

Currently all agents are running in one environment as the messaging system for remote communication is not yet realised. Thus the focus is left to the improvement of algorithms at the moment. As soon as the communication system is finished, the agents will run on separate devices and issues in coordination that may arise from network latencies could be considered.

References

1. Russel, S., Norvig, P.: Artificial Intelligence: A Modern Approach. 2nd Edition edn. Prentice Hall (2003)
2. Sesseler, R.: Eine modulare Architektur für dienstbasierte Interaktionen zwischen Agenten. PhD thesis, Technische Universität Berlin (2002)
3. Konnerth, T., Hirsch, B., Albayrak, S.: JADL - an Agent Description Language for Smart Agents. In Baldoni, M., Endriss, U., eds.: Declarative Agent Languages and Technologies IV. Volume 4327 of LNCS., Springer Verlag (2006) 141–155