Design and Development of Multi-Agent Systems in AGLOBE

Pavel Jisl, Michal Pěchouček

{jisl,pechouc}@labe.felk.cvut.cz

Gerstner Laboratory - Agent Technology Group
Department of Cybernetics, Czech Technical University

IEEE Distributed Human-Machine Systems 2008

Outline

- **AGLOBE** Introduction
- **AGLOBE** System Architecture
- Our First Agent
- Complex Scenario Demonstration
- Homework?

AGLOBE Introduction

- Multi-agent platform developed at Agent Technology Group (Gerstner Laboratory)
- `http://agents.felk.cvut.cz/aglobe/`
- `http://agents.felk.cvut.cz/aglobe/tutorial/`
- Application domains:
  - **simulation** of real-world environment and collective behavior of large communities
  - **scalability** - high number of fully fledged and fully autonomous agents
  - **agent migration** persistence and code and state migration
- integrates the concept of **agents** and services
- **Geographical Information System (GIS)** and **Environment Simulator (ES)**
- FIPA compliant on ACL level, but hasn't inter-platform communication

AGLOBE System Architecture

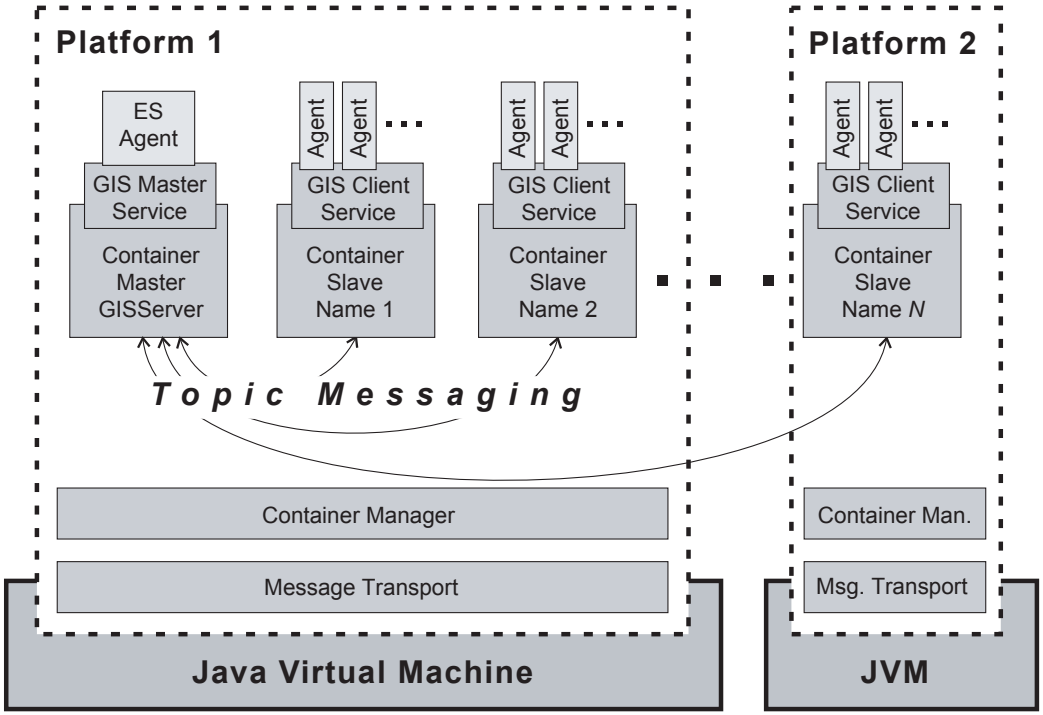


Figure 1: **AGLOBE** System Architecture Structure

- **Container Manager** takes care of starting, execution and finishing of containers
- **Message Transport** ensures an efficient exchanges of messages between two agent containers running in a single agent platform

Example 1: Running AGLOBE

1. Running **AGLOBE** without parameters
   - To run **AGLOBE** , type following to the command line:

     `java -jar AGlobe.jar`
   - **AGLOBE** help screen will appear, listing all parameters **AGLOBE** accepts
2. Running containers
   - To run **server** container, use following command

     `java -jar AGlobe.jar -name containername -server -gui`
   - To run **client** container, use following command

     `java -jar AGlobe.jar -name containername -client -gui`
   - To run only one container neither in server nor client mode (the location of agents in virtual space is not important)

     `java -jar AGlobe.jar -name containername -gui`
   - Note: *all containers must have unique names*
3. Running on different platforms (first platform is 0)
   - To run client **AGLOBE** container on different platform (different JVM), use

     `java -jar AGlobe.jar -name containername -gui -client -platform 1`

AGLOBE System Architecture – Agent Container

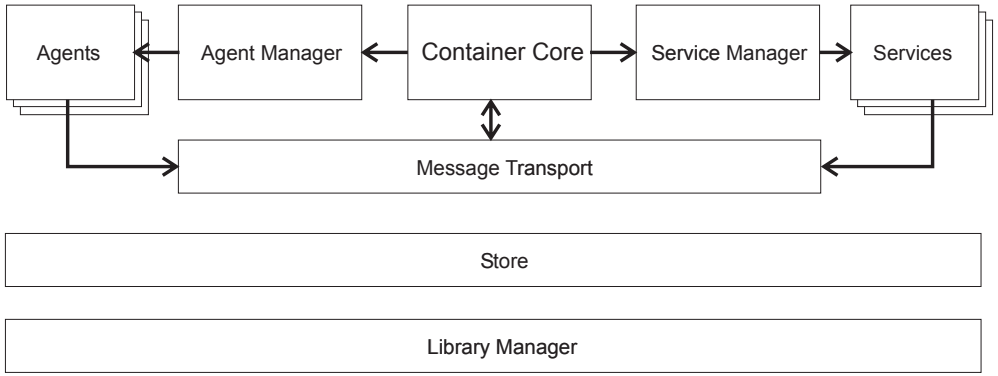


Figure 2: The Agent Container Structure

- **Container Core** starts up and shuts down all container components
- **Store** provides permanent storage for components
- **Library Manager** manages the libraries installed in the container
- **Message Transport** is responsible for sending and receiving messages from and to the container
- **Agent Manager** takes care of creation, execution and removal of agents on the container
- **Service Manager** starts and stops the services present in the agent container

AGLOBE System Architecture – Agent Container GUI

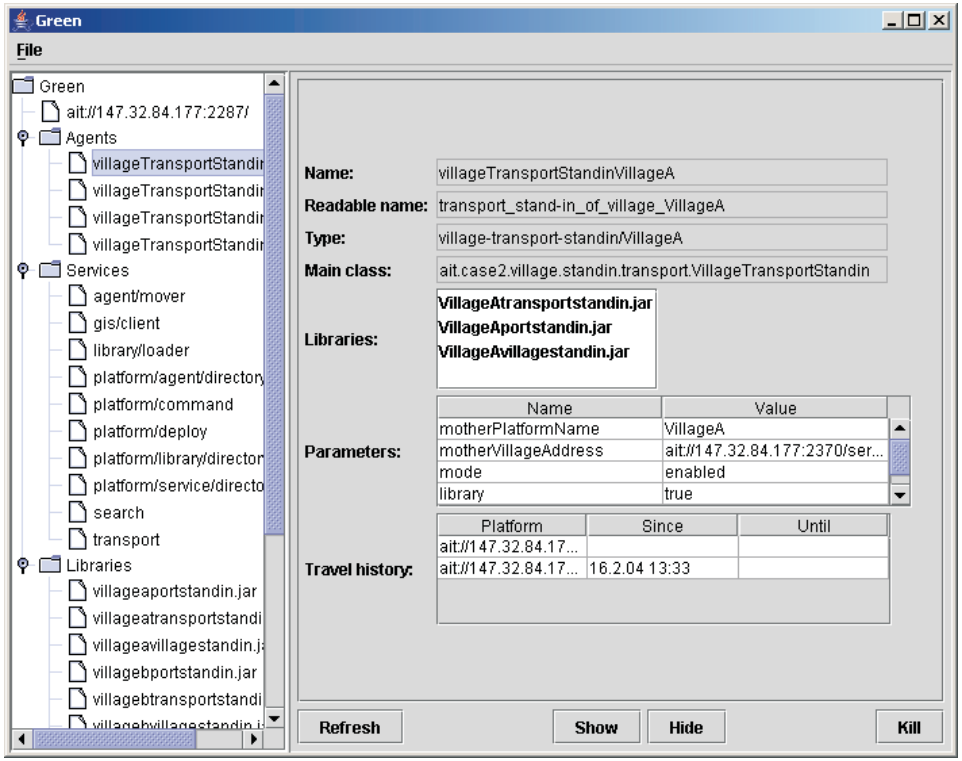


Figure 3: Container GUI: Agent Information

- provides easy way to inspect container state and to install or remove its components (agents, services, libraries)
- left side shows names of agents, services and libraries present on the container
- right side shows detailed information about the object selected in the tree

Example 2 : Starting and stopping agents

1. Starting agent from command line
   - To start `SendAgent` from command line, use following command:
     `java -jar AGlobe.jar -name container0 send0:examples.agent.send.SendAgent`
2. Starting agent using Agent Container GUI

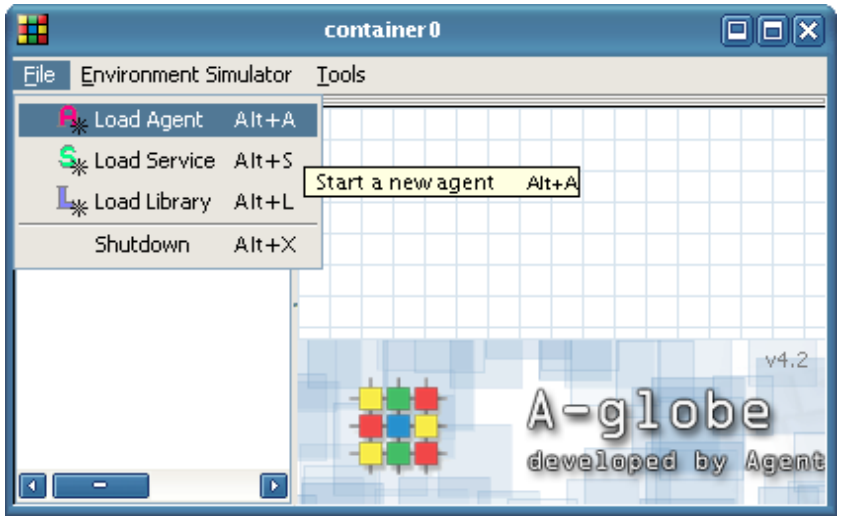


Figure 4: Container GUI

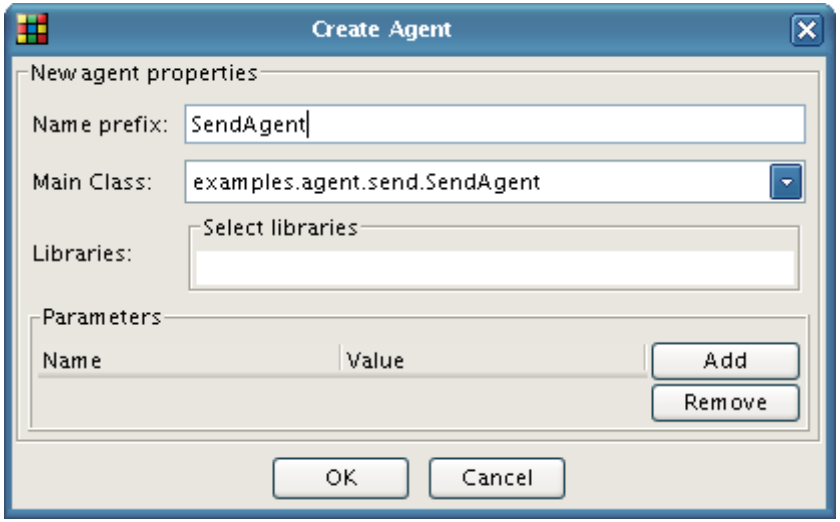


Figure 5: Create agent GUI

3. Stopping agent using Container GUI
   - Open Container GUI window and select the appropriate agent
   - Click on the `Kill` button located in the bottom right corner of the window

Example 3: Store – The permanent storage

- **store** is permanent storage and is used by all container components, agents and services
- Each entity has assigned its own virtual storage
- Each container maintains store with start up configuration and parameters of each agent
- Demonstration:
  1. Start container with parameter `-store` *`directory`*

     `java -jar AGlobe.jar -server -name container0 -gui -store store`
  2. Start some agent(s) using Container GUI
  3. Shutdown Container using `Alt+X` (or from menu)
  4. Directory `store` was created and start up configuration was stored
  5. Re-run container from commandline using step 1.
  6. All previously started agents will be started again
- Start up configuration is saved when the Container is shut down
- Old configuration is overwritten
- To avoid store changing, use parameter `-noStoreChange`

AGLOBE System Architecture – Message Flow

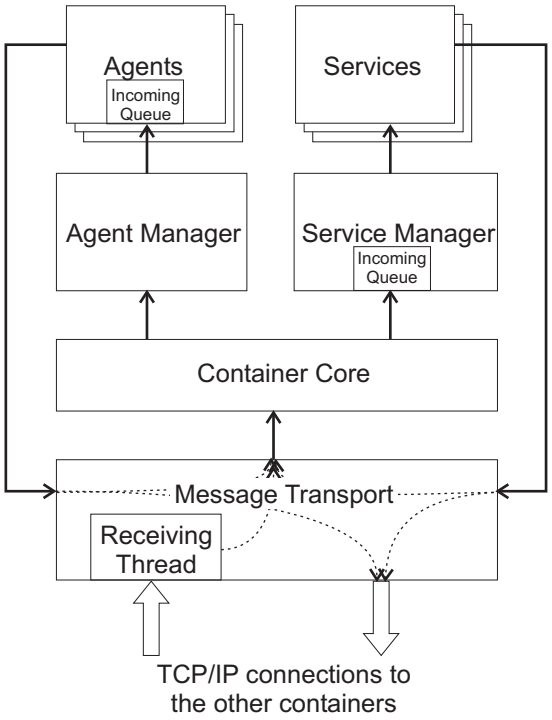


Figure 6: Message Flow

Message Transport

- Is responsible for sending and receiving messages
- Shared TCP/IP connection for message sending is created between every two agent containers hosted on different agent platform when the first message is exchanged between them
- Messages between two agent containers running in the same agent platform are passed via the platform-level *message transport* component
- The message structure respects FIPA-ACL
- Messages are encoded in Byte64 format
- Message content should be `Serializable`, better `Externalizable`

- Messages can be *unicast* or *multirecipient* (can be send using *multicast*)
- *Multirecipient* messages are send to all unique containers where recipient agents are running
- Message transport uses `tcp` (default) and `udp` protocols

AGLOBE System Architecture – Messaging

Message Sending

`sendMessage(m)`, where `m` is `aglobe.ontology.Message`

Message Receiving

- without conversation discrimination - all incoming messages processed in one method)
  - Agent must implement method:

    `public void handleIncomingMessage(Message m)`
- using **Conversation Manager** and **Tasks**
  - agent deals with multiple jobs simultaneously
  - *task* is able to send and receive messages and to interact with other tasks
  - *Conversation Manager* takes care of every message received to be routed to the proper task
  - Decision to which task a message should be routed depends on `ConversationID` ('reference number')
  - If agent needs to communicate with its own task (his own 2 tasks are sending messages to other), the local task must be registered to the other task – This solves situation, when both agent's task will have the same `ConversationID`.

Address and Messages

Agent/Service Naming and Addressing

- Agent name is globally unique and is generated by the platform during agent creation
- Service name is unique only within one agent container and is specified by the service creator
- `Address` syntax: `aglobe://platformIP:port/container_name/[agent|service]/name`

- Implemented in `aglobe.container.transport.Address`

  `String getHost()`, `String getPort()` returns IP resp. port

  `String getContainerName()` returns container name

  `String getName()` returns name of the agent or the service

  `boolean isAgent()`, `boolean isService()` returns true if it is agent's address resp. service address

Messages

- Implemented in `aglobe.ontology.Message`
- Message constants in `aglobe.ontology.MessageConstants`
- If target container is not accessible, `InvisibleContainerException` is thrown
- Multicast messages support - just add more receivers

Example 4: Simple Messaging

Unicast messages

1. Start at least two `SendAgent` (see Example 3)
2. Copy from `Send2` agent the parameter `name` (in `Sender` box) to clipboard
3. In `Send1` agent GUI fill the parameter `name` in `Receiver` box by pasting the clipboard content
4. Click the `Send` button in `Send1`'s GUI
5. `Send2` will report new received message in the `Incoming Message` box

Multicast messages

1. Start one (or more) `MulticastReceiver` agents

   Class `examples.agent.multicast.MulticastReceiver`
2. Start one (or more) `MulticastSender` agents

   Class `examples.agent.multicast.MulticastSender`
3. Try to send multicast message

AGLOBE System Architecture – Services

- Services run on each container
- Agents use services through *shells* – generic interface

GIS Service - Geographical Information System

- Used for `topic` messaging (system messages, doesn't depend on container accessibility)
- Agents subscribed for specific topic received it, when is submitted to their container
- **Topic message**
  - `String topic` contains the name of topic
  - `Object content` is serializable content (object) of topic
  - `String reason` is optional string field specifying reason for submitting topic
- **GIS Shell**

```
GISClientService.Shell gisShell = (GISClientService.Shell) getContainer().
getServiceManager().getService(this, GISClientService.SERVICENAME)
```

- **Topic subscription/unsubscription**

```
gisShell.subscribeTopic("SOME_TOPIC", this)
```

```
gisShell.unsubscribe("SOME_TOPIC")
```

System Architecture – Services (GIS cont.)

- **Topic handling**

  Implement `aglobe.service.gis.client.GISTopicListener` (Client)

  Implement `aglobe.service.gis.server.GISTopicServerListener` (Server)

  `handleTopic(...)` – executed when subscribed topic is received

- **Topic submission**

  Submitting agent located on *client container*

  `gisShell.submitTopicToLocal(...)` – topic for agents on *same container*

  `gisShell.submitTopicToServer(...)` – topic for agents on *server container*

  `gisShell.submitTopic(...)` – topic for agents on *same* and *server container*

  Submitting agent located on *server container*

  `gisServerShell.sendTopicToLocal(...)` – topic for agents on *same container*

  `gisServerShell.sendTopic(...)` – container name is specified as parameter

  `gisServerShell.broadcastTopic(...)` – topic for *any* container

AGLOBE System Architecture – Services II

Directory Service – "yellow pages"

- client container service responsible for distributing information about agents advertisements over all client containers
- **Directory Service Shell**

  `DirectoryService.Shell dsShell = (DirectoryService.Shell) getContainer().getServiceManager().getService(this, DirectoryService.SERVICENAME)`
- **Service advertisement**

  `dsShell.register(this,list)`, where `list` is list of services provided
- **Service subscription / unsubscription**

  `dsShell.subscribe(this,matchingFilter)`, `matchingFilter` interpreted as reg exp

  `dsShell.unsubscribe(list)`, `list` is list of services
- **Service handling**

  Implement `aglobe.container.sysservice.directory.DirectoryListener`

  `handleNewRegister(...)` and `handleDeregister(...)` – called when agent(s) registered service(s) resp. deregistered service(s)

  `handleVisible(...)` and `handleInvisible(...)` – called when registered agent(s) became visible resp. invisible (inaccessible)

AGLOBE System Architecture – Agent Migration and Cloning

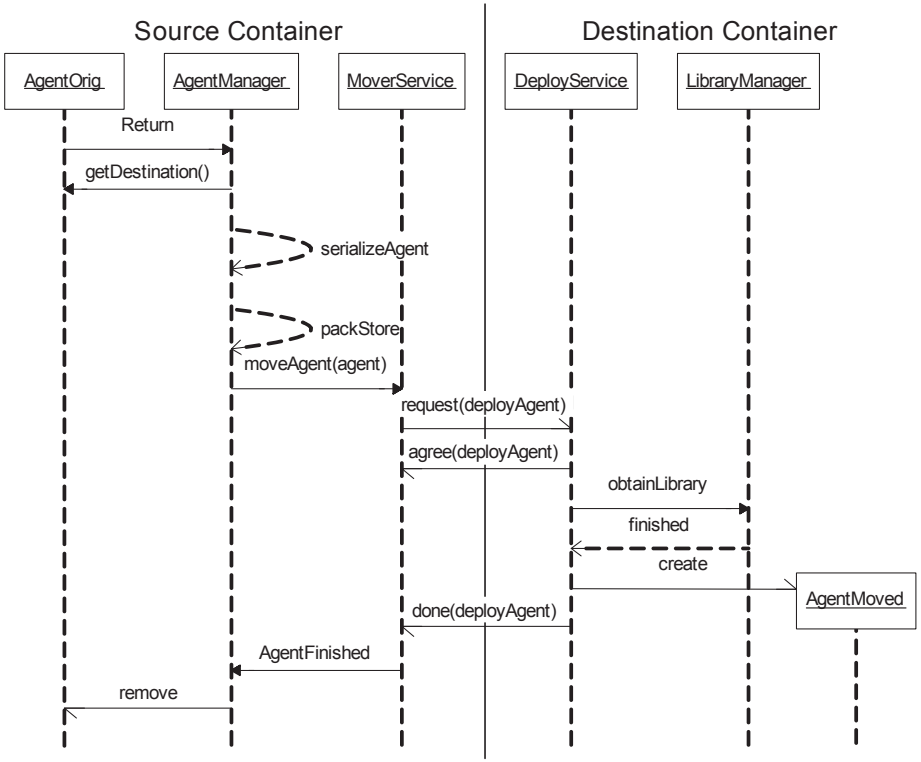


Figure 7: Agent Migration

- **AGLOBE** supports agent mobility and cloning
- All exceptions that might occur are properly handled
- Communication is secured by timeouts
- If migration cannot be finished, agent is re-created in its original container
- Agent Cloning

  same procedure, but the clone created on remote container can have different name and the original agent is not removed

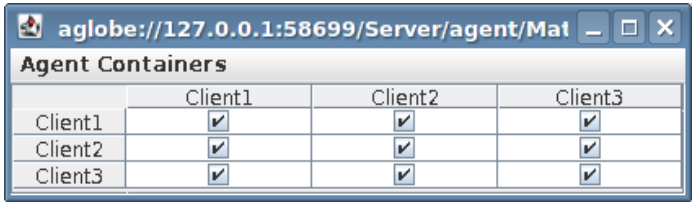
- `done` message
  - sent by the destination container but never received: two copies of agent emerge
  - received by the source container, but agent creation fails: agent is lost

Example 5: Agent Migration

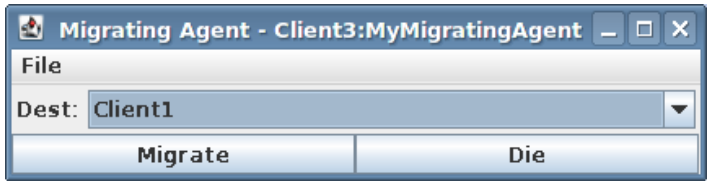
1. Start server and three client containers

```
java -cp AGlobe.jar;migratingagent.jar aglobe.platform.Platform -name Server -server -gui

java -cp AGlobe.jar;migratingagent.jar aglobe.platform.Platform -name Client1 -client -gui

java -cp AGlobe.jar;migratingagent.jar aglobe.platform.Platform -name Client2 -client -gui

java -cp AGlobe.jar;migratingagent.jar aglobe.platform.Platform -name Client2 -client -gui
```

2. On Server container, start `Matrix ES Agent` from menu `Environment Simulator`

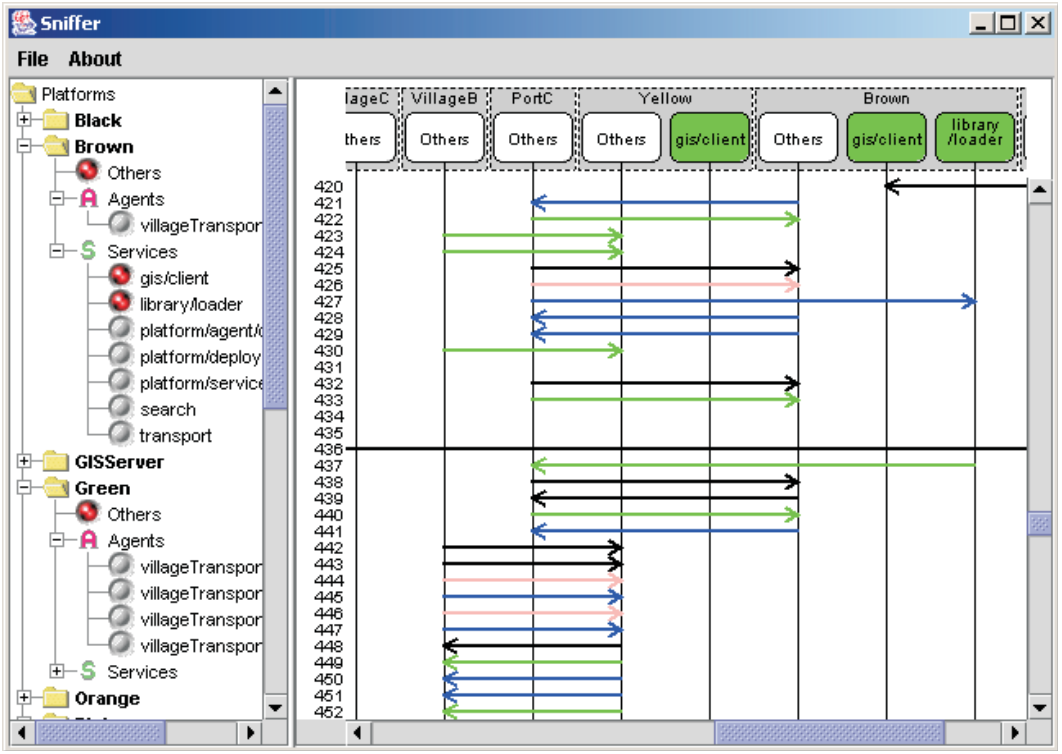


3. Start `MigratingAgent` on one of client containers (`examples.agent.migrating.MigratingAgent`)
4. Select destination container for migrating agent



AGLOBE Tools: Sniffer

- On-line monitoring for all messages and their transmission status (delivered or not-reachable target)
- Helps to resolve communication problems in system
- Runs only on server container



Our First Agent: The beginning

```
import aglobe.container.agent.Agent;
import aglobe.ontology.*;

public class FirstAgent extends Agent {

    public void init(AgentInfo ai, int initState) {
        System.out.println(this.getName() + " initialized with address "
            + this.getAddress().toString());
    }

    public void handleIncomingMessage(Message m) {
        System.out.println("Unexpected message received.");
    }
}
```

Our First Agent: Subscribing Topic

```
import aglobe.service.gis.client.*;
...
public static final String FIRST_TOPIC="OUR_FIRST_TOPIC";
...
public class FirstAgent extends Agent implements GISTopicListener {
...
     public void init(AgentInfo ai, int initState) {
...
          gisShell = (GISClientService.Shell) getContainer().getServiceManager().
               getService(this, GISClientService.SERVICENAME);
          if (gisShell != null) {
               gisShell.subscribeTopic(FirstAgent.FIRST_TOPIC, this);
          } else {
               System.out.println("GIS Client Service not found!  Maybe running on Server container.");
          }
}
     public void handleTopic(String topic, Object content, String reason) {
          if (topic.equals(FirstAgent.FIRST_TOPIC) { System.out.println("Topic received"); }
     }
```

Our First Agent: Directory Service

```
import aglobe.container.sysservice.directory.*;
...
public static final String FIRST_RECORD="OUR_FIRST_RECORD";
private LinkedList<Address> agents = new LinkedList<Address>();
...
public class FirstAgent extends Agent implements GISTopicListener, DirectoryListener {
...
     public void init(AgentInfo ai, int initState) {
...
          dsShell = (DirectoryService.Shell) getContainer().getServiceManager().
                    getService(this, DirectoryService.SERVICENAME);
          Collection<String> list = new ArrayList<String>();
          list.add(FirstAgent.FIRST_RECORD);
          if (dsShell != null) {
               try {
                    dsShell.register(this, list);
                    dsShell.subscribe(this, FirstAgent.FIRST_RECORD);
               } catch (DirectoryException ex) { ex.printStackTrace(); }
          }
```

Our First Agent: Directory Service – handling

Handling registration and deregistration

```
public void handleNewRegister(String containerName, DirectoryRecord[] records, String matchingFilter){
     for (DirectoryRecord dirRecord :  records) {
          if (!dirRecord.address.equals(getAddress())) {
               System.out.println(this.getName() + " New Service registered " + dirRecord.address);
               agents.add(dirRecord.address);
               sendWelcomeMessage(dirRecord.address);
               }
          }
}
public void handleDeregister(String containerName, DirectoryRecord[] records, String matchingFilter) {
     for (DirectoryRecord dirRecord :  records) {
          if (!dirRecord.address.equals(getAddress())) {
               System.out.println(this.getName() + " New Service deregistered " + dirRecord.address);
               agents.remove(dirRecord.address);
          }
     }
}
```

Our First Agent: Directory Service – handling cont.

Handling visibility

```
public void handleVisible(String containerName, DirectoryRecord[] records, String matchingFilter) {
     for (DirectoryRecord dirRecord :  records) {
          if (!dirRecord.address.equals(getAddress())) {
               System.out.println(this.getName() + " Service visible " + dirRecord.address);
               if (!agents.contains(dirRecord.address)) {
               sendWelcomeAgainMessage(dirRecord.address);
               }
          }
     }
}
public void handleInvisible(String containerName, DirectoryRecord[] records, String matchingFilter) {
     for (DirectoryRecord dirRecord :  records) {
          if (!dirRecord.address.equals(getAddress())) {
               System.out.println(this.getName() + " Service invisible " + dirRecord.address);
               if (agents.contains(dirRecord.address)) {
                    gisShell.submitTopic(FirstAgent.FIRST_TOPIC, "Not visible for container " +
                         this.getAddress().getContainerName());
               }
          } } }
```

Our First Agent: Messaging

```
public void handleIncomingMessage(Message m) {
     if (m.getPerformative().equals(MessageConstants.INFORM)) {
          if (m.getContent() instanceof String) {
               System.out.println("Message from " + m.getSender().getName() + " received " +
                    "with content " + m.getContent().toString());
          }
     } else if (m.getPerformative().equals(MessageConstants.QUERY)) {
          if (m.getContent() instanceof String) {
               System.out.println("Alive Message from " + m.getSender().getName() + " received " +
                    "with content " + m.getContent().toString());
          }
     } else {
          System.out.println("Unexpected Incoming Message");
     }
     m.release();
}
```

Our First Agent: Messaging cont.

```
private void sendWelcomeMessage(Address address) {
     try {
          Message m = Message.newInstance();
          m.setReceiver(address);
          m.setSender(this.getAddress());
          m.setPerformative(MessageConstants.INFORM);
          m.setContent("Welcome in agents world");
          sendMessage(m);
     } catch (InvisibleContainerException ex) {System.out.println("Invisible Container Error"); }
}
private void sendWelcomeAgainMessage(Address address) {
     try {
          Message m = Message.newInstance(MessageConstants.INFORM, this.getAddress(), address);
          m.setContent("Welcome back");
          sendMessage(m);
     } catch (InvisibleContainerException ex) { System.out.println("Invisible Container Error");}
}
```

Our First Agent: Simple Timer Task

```
public void init(AgentInfo ai, int initState) {
...
     TimerTask sendAliveMessage = new TimerTask() {
          public void run() {
               Message m = Message.newInstance(MessageConstats.QUERY);
               m.setSender(getAddress());
               m.setContent("Alive Message from " + getName());
               for (Address address :  agents) {
                    m.setReceiver(address);
                    try {
                         sendMessage(m);
                    } catch (InvisibleContainerException ex) {
                         System.out.println(getName() + " Agent " + address.getName() + " is not visible");
                    }
               }
          }
     };
     // schedule sendAliveMessage with 0 seconds delay and repeat each 5 seconds
     getContainer().TIMER.schedule(sendAliveMessage, 0, 5000);
...
```

Our First Agent: Simple Timer Task – Multicast solution

```
public void init(AgentInfo ai, int initState) {
...
     TimerTask sendAliveMessage = new TimerTask() {
          public void run() {
               Message m = Message.newInstance(MessageConstats.QUERY);
               m.setSender(getAddress());
               m.setReceivers(agents);
               m.setContent("Alive Message from " + getName());
               try {
                    sendMessage(m);
               } catch (InvisibleContainerException ex) {
                    System.out.println(getName() + " Agent " + address.getName() + " is not visible");
               }
          }
     };
     // schedule sendAliveMessage with 0 seconds delay and repeat each 5 seconds
     getContainer().TIMER.schedule(sendAliveMessage, 0, 5000);
...
```

Our First Agent: Running

1. Start Server container and start Matrix ES Agent and Sniffer agent
2. Start Client container 1, our First agent and look into console and Sniffer agent
3. Start Client container 2, our First agent and look into console and Sniffer agent
4. Try to uncheck visibility between containers 1 and 2 and look into console and Sniffer agent
5. Start another Client container
6. Stop one container
7. Shutdown all containers

Demonstration of complex scenario

Homework?

- `http://agents.felk.cvut.cz/aglobe/`

  Latest version of **AGLOBE**

  JavaDoc documentation

  AGLOBE Manual

  Quick start document (outdated)

  This tutorial

- Contact us

  Michal Pěchouček (`pechouc@labe.felk.cvut.cz`) – Head of ATG

  David Šišlák (`sislakd@fel.cvut.cz`) – Head of **AGLOBE** development

  Pavel Jisl (`jisl@labe.felk.cvut.cz`)