living agents CE Quick Start

Version 3.2

2001-11-26

Contents

1 Overview

1.1 About This Document

This document gets you started with the **living agents** toolset. You will be introduced to all tools and components you need in order to use **living agents**. Such tools and components are the **living agents** Runtime System (LARS) the **living agents** Control Center (LACC) the **living markets** Development Suite (LMDS) and the **living markets** Demo Application (LMDA). You will be guided through your first steps making, running, and controlling agents.

This document is not meant as a reference for **living agents** or any of the other components mentioned herein, but rather as a means to get you (quickly) started. For detailed reference see the documents listed below.

1.2 Documentation Structure

livingMarkets_AgentHandbook_CEV3.2: This Handbook gives an overview of the living market agents, of LARS (living agents runtime system), of the LARS cockpit, of the messaging and the communication within the system. It shows in detail which agents exist, what every agent does and what they can be used for.

The main parts of the Handbook are:

- LIVING AGENTS RUNTIME SYSTEM
- MESSAGING
- ERROR MESSAGING
- AGENTS
- AGENT-LIKE CLIENTS
- LOGIC CONTROL

- PLATFORM SYNCHRONIZATION
- AGENT-RELATED CONFIGURATION

livingMarkets_QuickStartV3.2 : This Quick Start guides you through the main components of **living markets** that you need in order to build your fist agent.

2 Installation

The living agents toolset is shipped with an installer for Windows platforms and a tar file for Linux. Both contain the living agents toolset and a Java Runtime environment.

2.1 Installation on Windows

To install living agents on a Windows system simply run the file

```
Setup_livingagents.exe
```

and follow the instructions of the installer[1].

After installation you find a **living agents** program group folder on your desktop (Figure 2.1). The start icons for the various tools can be found in the sub-directory `living agents\images` of your installation directory.

[1] Make sure you install living agents in a new folder since the de-installation will remove the complete installation directory.

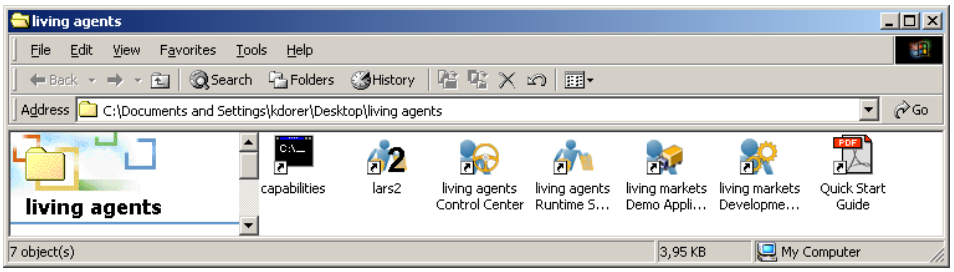

Figure 2.1: The living agents program group on Windows.

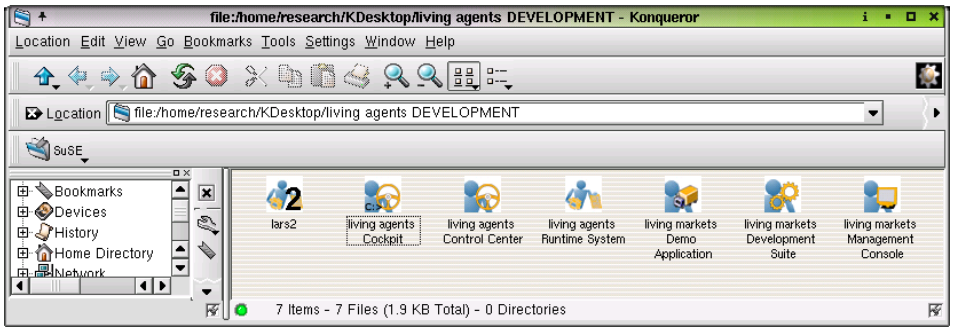

Figure 2.2: The living agents program group on Linux.

2.2 Installation on Linux

To install living agents on Linux you have to do the following steps:

1. Copy the file `Setup_livingagents.tar` to an empty directory and change to that directory.
2. Untar the file `Setup_livingagents.tar` using the command

   ```
   tar -xvf Setup_livingagents.tar
   ```

3. Run the installation script of the installed bin directory

   ```
   bin.Linux/install.sh
   ```

4. Run the environment script of the installed bin directory

   ```
   . bin/setEnvironment.sh
   ```

   We suggest to call this configuration script from your personal startup file, e.g. `.profile` or `.bashrc`.

The start icons for the various tools can be found in the sub-directory `$InstDir\Toolbar` of your installation directory. (Figure 2.2).

3 living agents Runtime System

The **living agents** Runtime System (LARS) is a scalable agent server and thus the runtime environment for the agents. LARS provides the infrastructure for running agents and communication between agents. Thereby the following protocols are supported:

- Sockets
- Secure Sockets
- RMI
- HTTP
- HTTPS
- JMS

It is possible to integrate 3rd party application servers into the agent server LARS in order to make use of capabilities running on such application servers.

3.1 How to Start LARS

In order to start LARS simply click the icon 

. A command window will pop up and show the log of LARS.

3.2 The LARS Start Log

On starting LARS, the following output is shown. This is an extract of the full LARS log.

First the initial setup is shown:

```
PROJECT_HOME is .
JAVA_HOME is .\jre1.3.1
path is .\jre1.3.1\bin;.\bin

living agents(c) runtime system 3.1.3
logFile='System.out', logLevel='info', globalLogPath='log'
setting JVM default locale to 'en_US'
setting platform ID to 'lars'
parameter 'agentManagerLogType' (the log type of the AgentManager)
not given. Using default value 'LARS'
going to use local IP address 'localhost'
globalConfigPath 'conf' is not an absolute path!
going to use encoding 'UnicodeBig'
parameter 'useSecurityPolicyFile' is 'off', using our own security
manager only
```

Thereby it is important to note the platform ID "`lars`" and the IP address "`localhost`". We will need these parameters later on.

Next the system boots through the various run levels:

```
run level CONFIG_FILE_INTERPRETED
run level SECURITY_MANAGER_SET
run level LOCAL_IP_ADDRESS_KNOWN
run level MESSAGE_ROUTER_CREATED
run level RUNNING
```

The system agents are started:

```
agent 'amr' started
agent 'aps' started
agent 'asi' started
agent 'asl' started
agent 'ajsl' started
agent 'ass' started
agent 'asch' started
```

The application agents are started:

```
agent 'AgentUserInterface' started
agent 'AgentDriveSimulation' started
```

Last you are told which listener might be connected via which port:

```
Socket listener is waiting for connections on port: 8005
JSocket listener is waiting for connections on port: 8004
```

This information will be needed later on, too.

May be some of the startup messges are shown in a different order. This depends on the actual non-deterministic start order of the agents.

Now LARS is running on your local computer.

4 living agents Control Center

The **living agents** Control Center (LACC) is a management tool to control LARS and the agents running on it. This control is executed via a graphical user interface[1]. LACC provides an extensive set of commands to monitor, debug, and manage the system. Additionally, you may configure your own commands. LACC is a graphical user interface to the command line tool **living agents** Cockpit (LAC). Thus LACC serves the same purpose than LAC.

By using LACC you act as an agent running on the platform. This way you may interact with other agents, and you view the platform as an agent.

In the following you will learn how to use the most important components of LACC and you will do the first steps to act as an agent on your platform via LACC.

Before you continue make sure that the LARS agent platform is started on your local computer (see section 3.1).

4.1 How to Start LACC

To start LACC, click the icon 

. The application starts up and automatically connects to the local LARS.

[1] There is also a command line tool **living agents** Cockpit available, which is, however, not part of living agents CE.

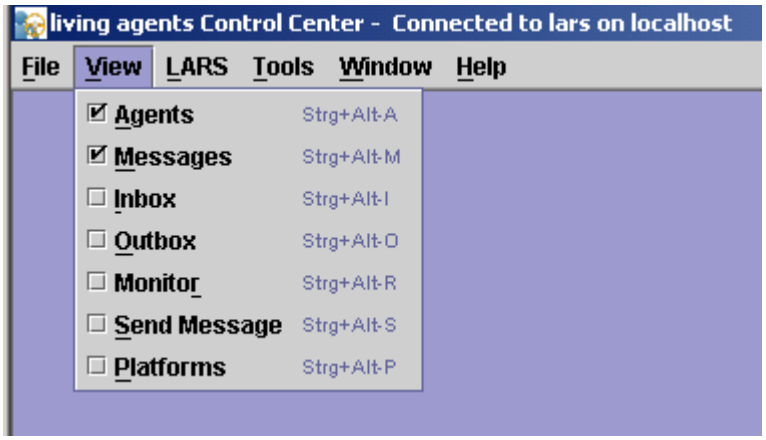

Figure 4.1: The View menu of LACC.

4.2 Most Important LACC Components

We will introduce you here only to the most important components of LACC, which you need for a quick start. These components are

Active Agents A list of all currently active agents running on your LARS systems.

Messages A list of preconfigured messages, which can be sent to agents.

Outbox A list of all messages you sent to agents.

Inbox A list of all messages you received from agents.

A more detailed description of these components follows in Sub-Sections 4.2.1 - 4.2.4 below.

To show or hide the various components of LACC, select or de-select them in the View menu as shown in Figure 4.1.

4.2.1 List of Active Agents

The list of active agents is shown by default after connection to LARS. It shows all currently running agents on the LARS platform you are connected to. By default, these are the system and application agents that are started automatically by LARS (see Sub-Section 3.2). Figure 4.2 shows this situation.

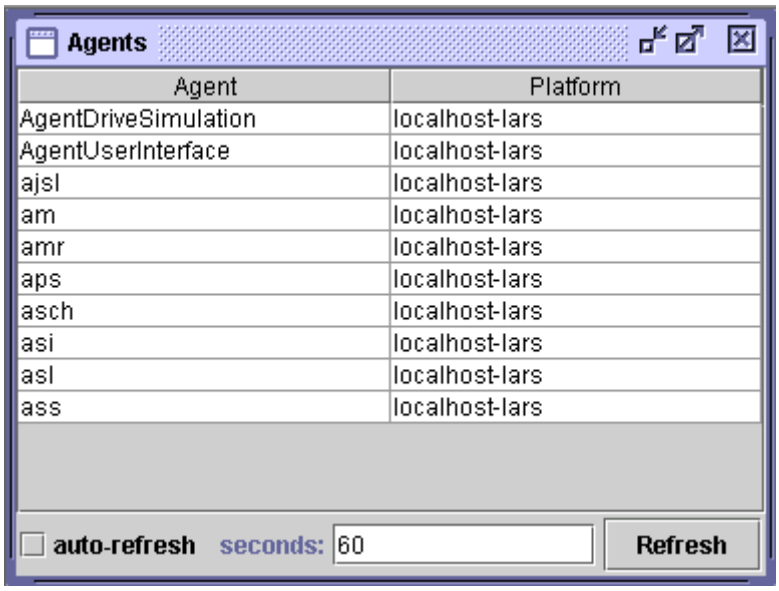

Figure 4.2: The list of active agents after starting LARS, displaying all agents started automatically by LARS.

The list of active agents is composed of two columns. The first one shows the name of the agents, the second one the name of the platform and the LARS ID (separated by an hyphen "-") the respective agent is running on.

You may refresh the list by clicking the button "Refresh". If you want the list to automatically refresh, set the refresh period in the box "seconds:" and click the checkbox "auto-refresh". The default refresh period is 60 seconds.

4.2.2 Messages

The window for the messages component is shown in Figure 4.3. This window consists of the following components:

List of Message Files: The list of all preconfigured files containing messages is shown in the top left frame in form of a folder tree. You may add a message file by adding an XML file with the extension `.msg` to the folder

```
$InstDir\conf\cockpit\messages
```

or a sub-folder thereof.

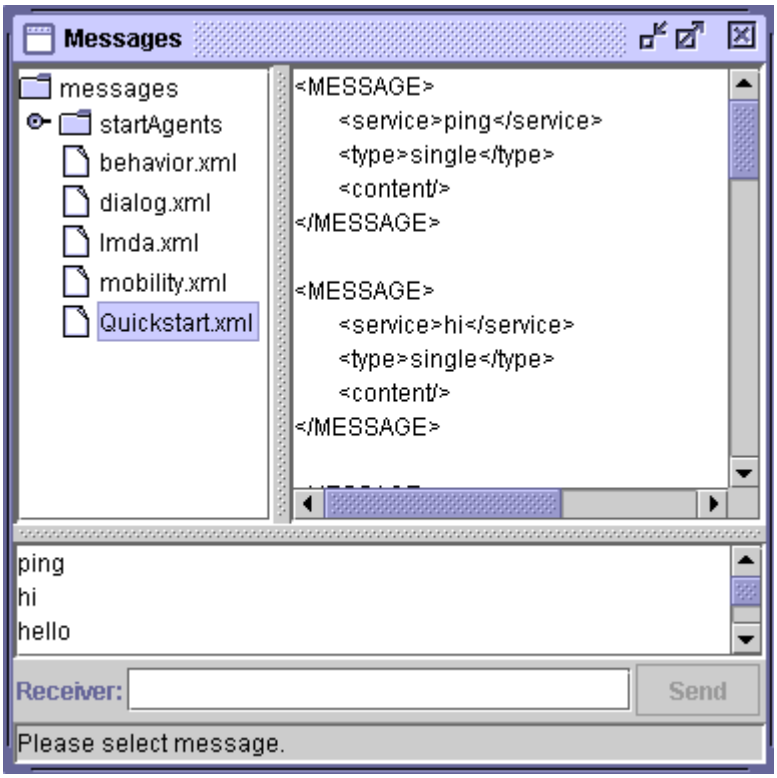

Figure 4.3: The Messages window of LACC.

XML Message: If you select a message file from the list of message files in the top left frame, the corresponding XML file is displayed in the top right frame. The XML file may contain multiple messages. For each message it is mandatory to name the service to be called by the message and the message type. Optionally you may attach a label to the message and provide content to the message.

List of Services: The field below the two top frames provides a list of all services contained in the selected message file. Thereby the services are either listed by their service name, or, if a label has been attached to a message, the service is listed by that label. If you send a message to an agent, you call the correspondent service of the agent.

Receiver Field plus Send Button: Here you may enter the agent to which you want to send the message you previously selected in the List of Services. The agent must be entered in the form

```
<agent_name>@<platform_name>-<LARS_name>
```

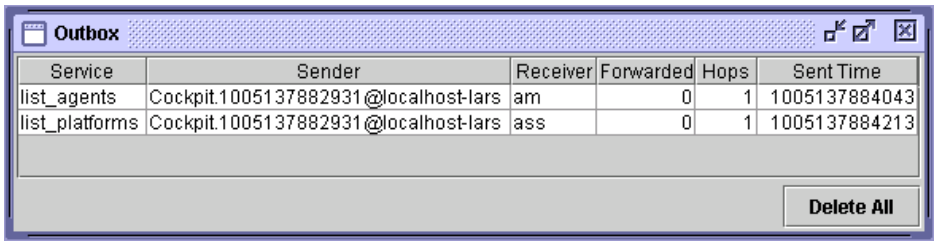

Figure 4.4: The Outbox window of LACC, containing the messages being sent on starting LACC.

Alternatively you can click on an agent in the agent list. The receiver field is automatically filled. The `send` button sends the selected message to the agent in the receiver field. A message can also be sent by dragging it from the list of services and dropping it on an agent in the agent list[2].

4.2.3 Outbox

Figure 4.4 shows the LACC outbox with the messages being sent on starting the LACC. The outbox consists of the following columns:

Service The service which is called by the message.

Sender The sender of the message. The sender may be an agent, the LACC or the LAC.

Receiver The receiver of the message. The receiver may be an agent, the LACC or the LAC.

Forwarded How often the message was forwarded.

Hops The number of hops it took the message to reach the receiver.

Sent Time The date and time the message was sent.

If the LACC (or LAC) was the sender or receiver of a message, you see that the sender or receiver is named "Cockpit". To the name "Cockpit" a lengthy number is attached. This number is needed in order to create a

[2]The drag&drop does not work if the LACC window was not active before, because in this case the mouse click only activates the window. Thus if the LACC window is inactive, activate it first e.g. by clicking with the mouse somewhere on the window, and then drag&drop the message you want to send.

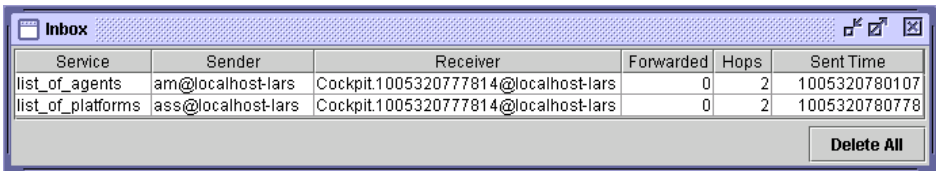

Figure 4.5: The Inbox window of LACC, containing the messages being received on starting LACC.

Figure 4.6: The "LARS" menu list of LACC.

unique name for the cockpit. In Figure 4.4 you see this, as the cockpit was the sender of the two messages in the outbox.

With the button "Delete All" you can empty the outbox.

4.2.4 Inbox

Figure 4.5 shows the LACC inbox with the messages being received on starting LACC. The inbox consists of the same columns as the outbox. With the button "Delete All" you can empty the inbox.

4.2.5 Execute Command Frame

In order to open the Execute Command Frame, select menu `LARS>Execute` (see Figure 4.6). The Execute Command Frame is shown in Figure 4.7.

In the Execute Command Frame you may enter commands, which are supported by LACC, into the field "Command:". This field serves as a command prompt as in the command line version LAC. You can find a list of all such commands in the drop down list "Available Commands". If you select a command from this list, it is entered into the field "Command:". To execute

Figure 4.7: The Execute Command Frame, with the command "availablecommands" being selected.

the command, which has been entered into the field "Command:", press the button "Execute"[3].

Note: Currently the list "Available Commands" contains all commands, which are planned to become supported by LACC. However, at this moment only some of them are implemented!

4.3 Exercise

The first exercise is to drag&drop the message "ping" to the agent "AgentUserInterface". To do this, do the following:

1. Activate the LACC window.
2. Make sure the messages window is open (`View>Messages`) and the `Quickstart.xml` message file is selected.
3. Click onto the message "ping" in the List of Services of the Messages window.
4. Move the mouse to the agent "AgentUserInterface" in the List of Active Agents while you hold the mouse button pressed.
5. Release the mouse button.

As a result of this action, you will see the message "ping" you sent to the agent "AgentUserInterface" in the Outbox, and the message "pong" the agent "AgentUserInterface" sent to you as an answer in the Inbox. Figure 4.8 shows the situation in the LACC window.

[3]For some commands the results may not be directly visible.

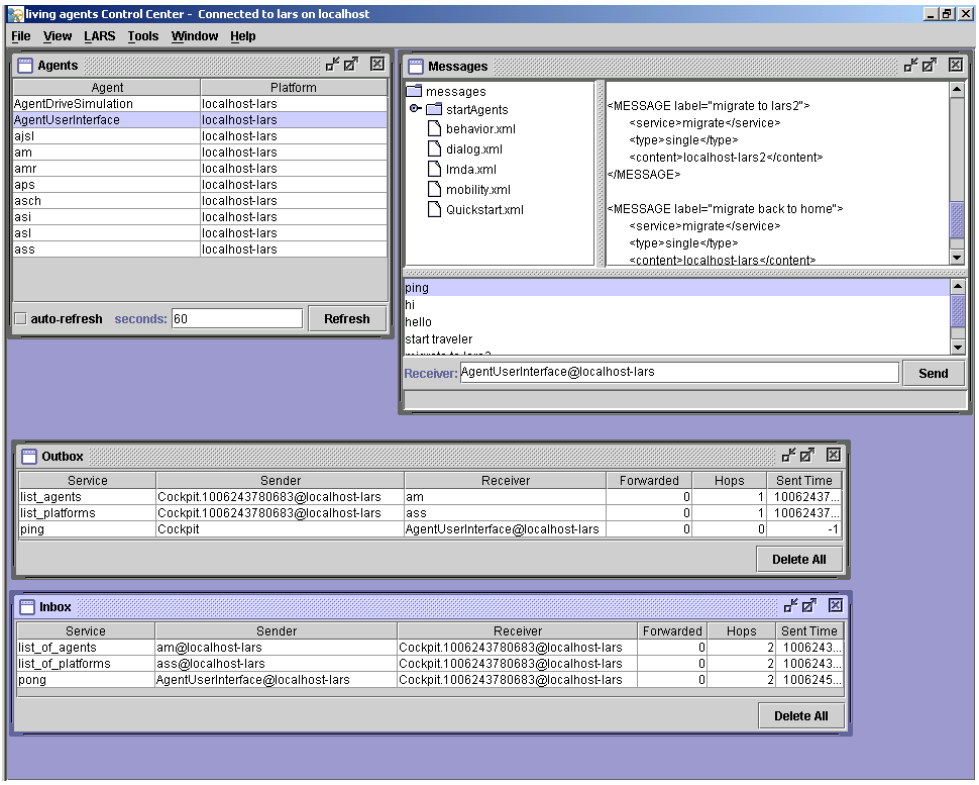

Figure 4.8: The LACC window after it looks after successfully sending the message "ping" to the agent "AgentUserInterface" via drag&drop.

As you see in Figure 4.8, the field "Receiver:" in the Messages window has been filled in with the agent "AgentUserInterface". Send the message "ping" again by pressing the button "Send", and you get the messages "ping" and "pong" again in your Outbox and Inbox, respectively.

Next we send the message "ping" to the same agent via the Execute Command Frame. To do this, you have to type the command "ping" and separated by a blank the address of the receiving agent into the field "Command:" of the Execute Command Frame. Execute the command via the button "Execute". The situation in the LACC window afterwards is shown in Figure 4.9.

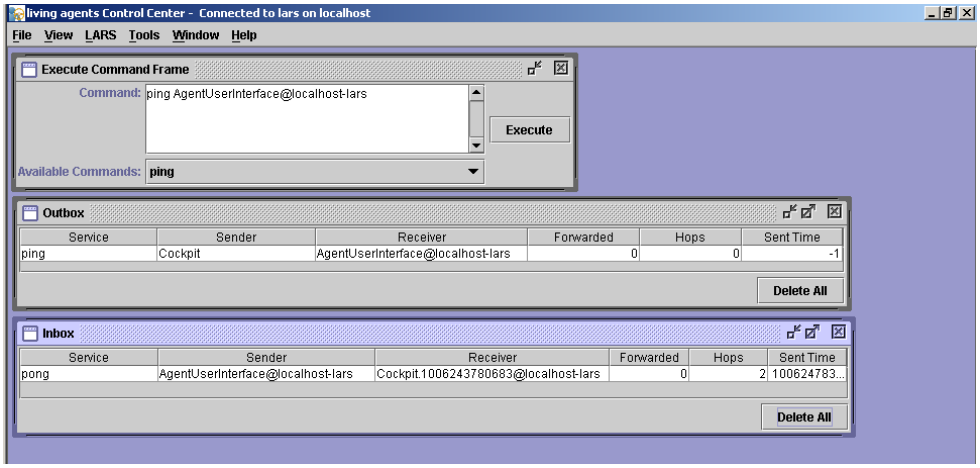

Figure 4.9: The LACC window after successfully sending the message "ping" to the agent "AgentUserInterface" via the Execute Command Frame.

5 living markets Development Suite

The **living markets** Development Suite LMDS is the visual tool for agent-oriented design of applications. It allows to design agent scenarios, which define the agents and their interaction in the application. The LMDS generates agent configuration files from the designs that can immediately run on a LARS platform.

In this chapter we explain how to create and test an agent using LMDS. We then describe how communication between two agents can be achieved. The third part is a detailed explanation on how to create mobile agents. We show how to allow them to move between different LARS platforms on a single computer and how agents can migrate between different computers. Finally we explain how to create the basic building blocks of agents: perception and action capabilities.

5.1 How to Start LMDS

Before starting the **living markets** Development Suite, the LARS should be running (see Chapter 3). To start LMDS, click the icon 

. It will start with an empty workspace.

5.2 A Simple Agent

In this section we describe the steps that are necessary to create a single agent in the LMDS. The agent's only task will be to reply to a message by

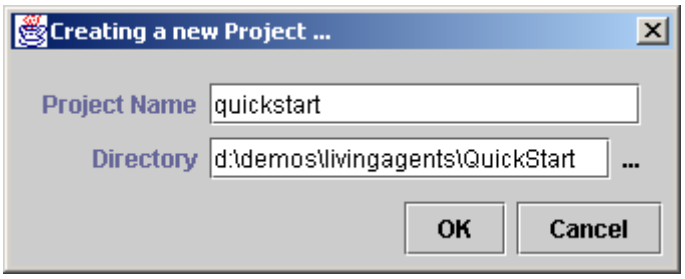

Figure 5.1: New Project dialog.

displaying a dialog box. The description contains how to create, start, test and stop the agent.

5.2.1 Create a New Project

To start working with LMDS, a new project has to be created. This can be done by clicking on the 'New Project' tool button  . In the new project dialog (Figure 5.1) you have to specify the name of the project as well as the directory where the project files should be located. We choose the name 'quickstart' and the `Quickstart` folder of the installation directory of living agents. After pressing OK, you are prompted for the name of a new scenario within your project. The scenario will later contain the agents and communication links. We choose the name `quickstartDemo`. After pressing OK, an empty scenario window appears.

5.2.2 Create an Agent

Now we create a new agent by right clicking on the scenario window and choose `mobile agent` to create an agent that can travel between platforms. In our example we create a mobile agent with name "Snoopy". A (yet empty) overview of the agent is displayed in the scenario window (see Figure 5.2).

5.2.3 Specify an Agent's Business Logic

The next step is to specify the business logic of the agent. This is done by clicking on the properties tool button of the agent dialog  . In the agent properties dialog (see Figure 5.3) all the information of an agent can

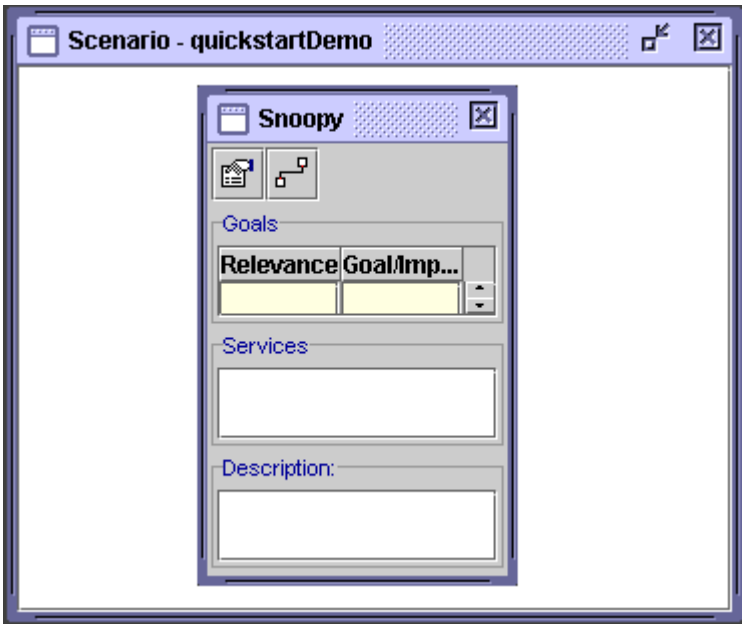

Figure 5.2: Agent scenario with a single (yet unspecified) agent.

be configured including its business logic.

First you have to select in the uppermost drop down box which type of agent logic the agent should use. Two agent logics are currently supported: a reactive service logic and a proactive extended behavior network. For our simple demo agent we choose the service logic. Previously entered business logic may be lost when changing between different agent logics. Therefore a confirmation dialog reassures that the change of agent logic is intentional. Since no business logic was entered so far, we can confirm the change of agent logic.

The agent properties dialog shows four panes: Basics, Capabilities, Logic Setup and Services.

Basics: In the 'Basics' pane you can specify the name of the agent and optionally an icon file and a description for the agent (see Figure 5.3).

Capabilities: The 'Capabilities' pane lets you choose from the capabilities (containing actions and perceptions) an agent can incorporate. You can press the Add button to add new capabilities to the agent (see Figure 5.4). The 'Add capabilities' dialog appears (see Figure 5.5) with a drop-down list of existing capabilities. For our example we choose the 'Swing' capabilities that contains actions to display message boxes.

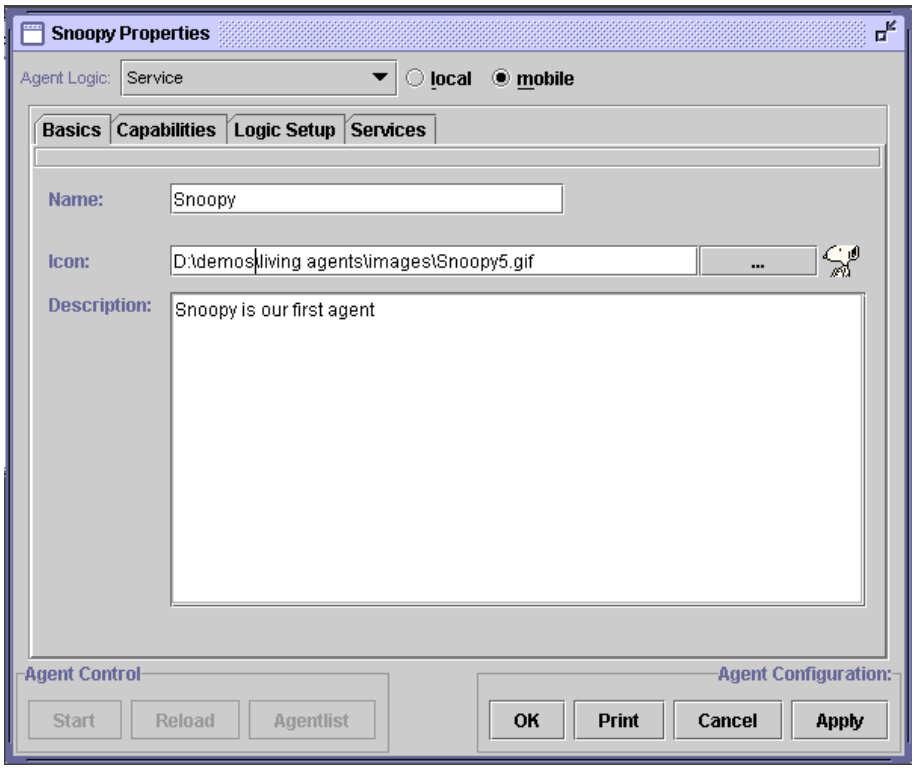

Figure 5.3: Basic properties of an agent.

Logic Setup: The 'Logic Setup' pane allows advanced configuration of the logic engine in XML format. For this example no special logic engine configuration is necessary.

Services: You can specify the business logic of a service agent in the 'Services' pane (see Figure 5.6). Initially, no service is defined for the agent. To add a new service press the 'New Service' button of the agent properties dialog  and provide the name `hello` for this new service in the 'Add Service' dialog. A service is performed once the agent receives a message requesting the service. Our example service shall just display a message box. To achieve this we have to add an action to the service. This is done by the following steps:

1. Click on the hello service in the `Services` window of the services pane of the agent properties window. The `Service Tree` now displays the hello service tree with a `Sequence` and a `Exception` branch.
2. Switch to the capabilities dialog with menu `Window>Capabilities`.

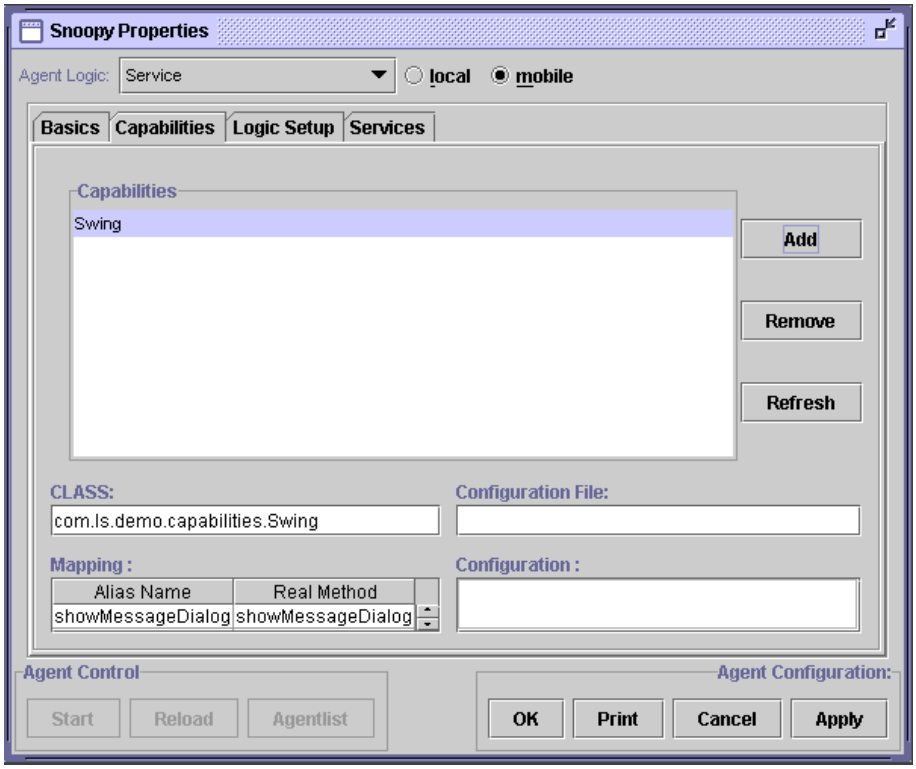

Figure 5.4: Capabilities properties of an agent.

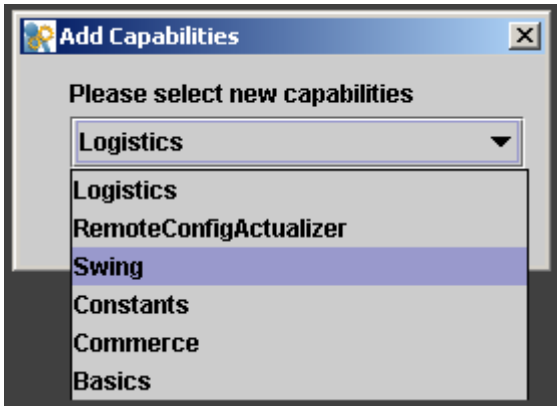

Figure 5.5: Add capabilities dialog.

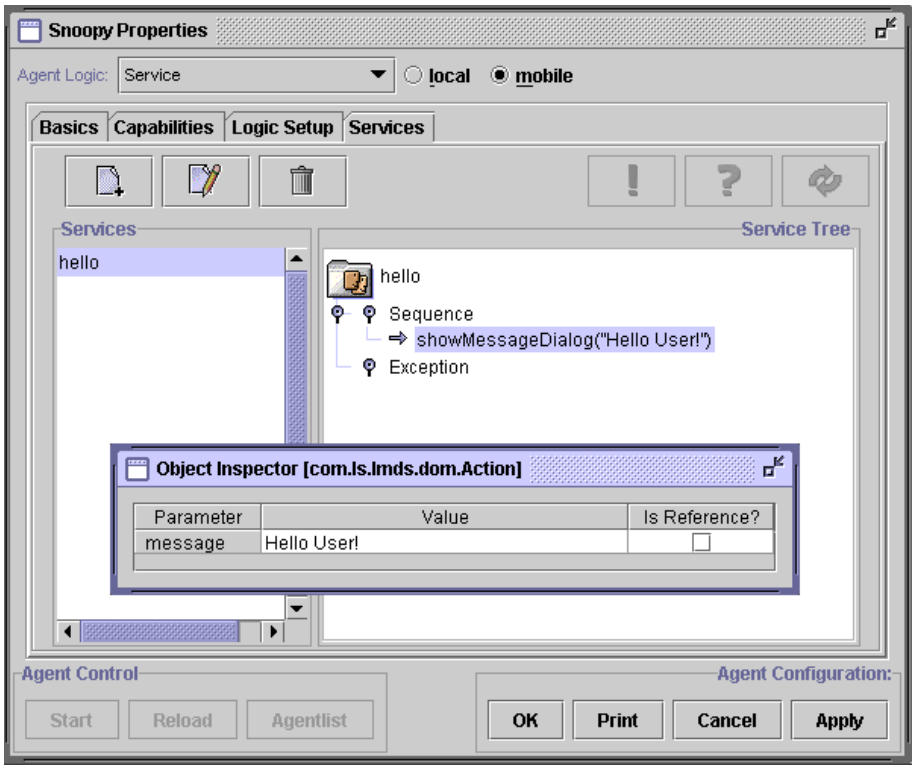

Figure 5.6: Services of the example agent and Object Inspector.

3. Expand the capabilities tree by clicking the icon  (see Figure 5.7) of the LMDS toolbar.
4. Drag the `showMessageDialog` action of the capabilities dialog to the service tree window of the agent properties dialog and drop it onto the `Sequence` branch.

You can specify the content of the message by clicking on the 'showMessageDialog' action in the agent properties window. If the Object Inspector window is not visible, select menu `Window>Object Inspector`. The Object Inspector window will display a table with all parameters this action needs. In our example we just have to specify a text, e.g. "Hello User!", for the message parameter and confirm with `<RETURN>`. The agent properties window will display the text as parameter of the 'showMessageDialog' action (see Figure 5.6).

Finally you have to apply the settings by pressing the `Save` button of the agent properties dialog. You also should save the scenario now by pressing

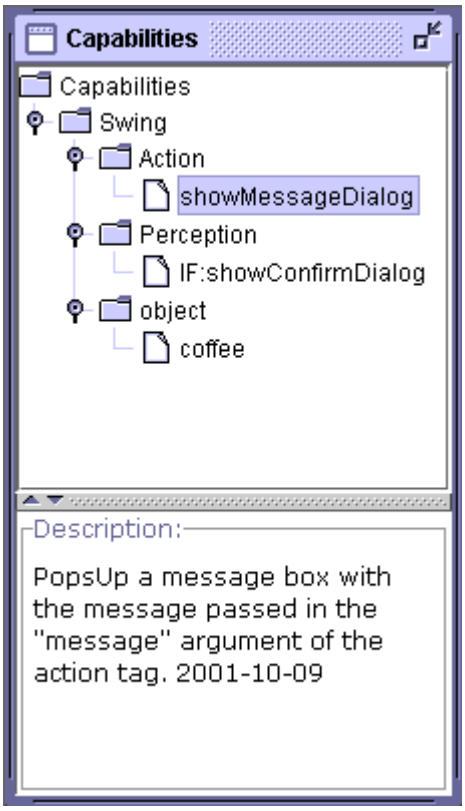

Figure 5.7: Capabilities tree of our example agent.

the `Save Project` button  of the LMDS toolbar.

5.2.4 Start an Agent from LMDS

To start the example agent on the LARS system you first have to connect the LMDS to LARS by clicking on the connect button  . After successful connection the window title will display the connection status. Also, the 'Agent Control' buttons in the agent properties window will be enabled. You can now start the agent by clicking on the 'Start' button in the 'Agent Control' field. By pressing the 'Agentlist' button you can display a list of agents running on LARS. The list contains Snoopy (Figure 5.8).

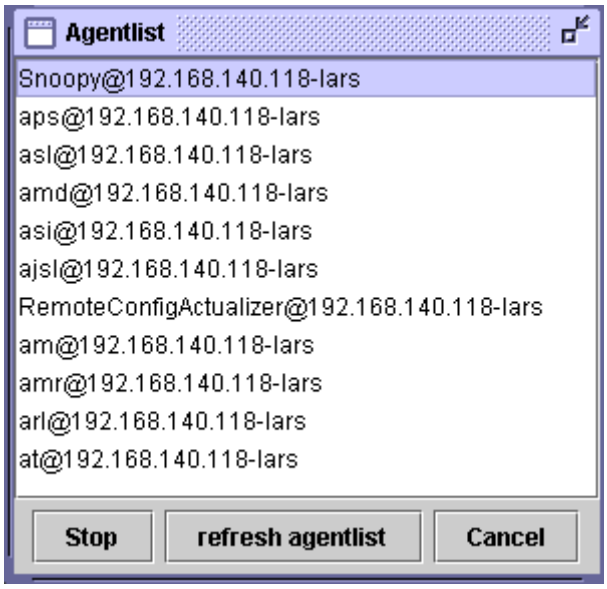

Figure 5.8: List of agents running on the LARS platform.

5.2.5 Test an Agent with LACC

To test the agent you have to switch back to LACC. Press the `Refresh` button of the agents window to see Snoopy. Open the messages window and select the `Quickstart.xml` message file. Select the `hello` message, drag and drop it to Snoopy in the agents window (Figure 5.9). This sends a `hello` message to Snoopy triggering the `hello` service of Snoopy. A message box with text `Hello User!` appears (Figure 5.10).

You can now close the agent properties dialog in the LMDS by clicking the `OK` button.

5.3 Communication Between Agents

For an inter-agent communication you need a second agent that exchanges messages with Snoopy. Therefore, you have to switch back to the LMDS.

5.3.1 Create a Second Agent

We name our second agent "Woodstock". You create "Woodstock" as described in Sub-Section 5.2.2 with following differences. In the capabilities

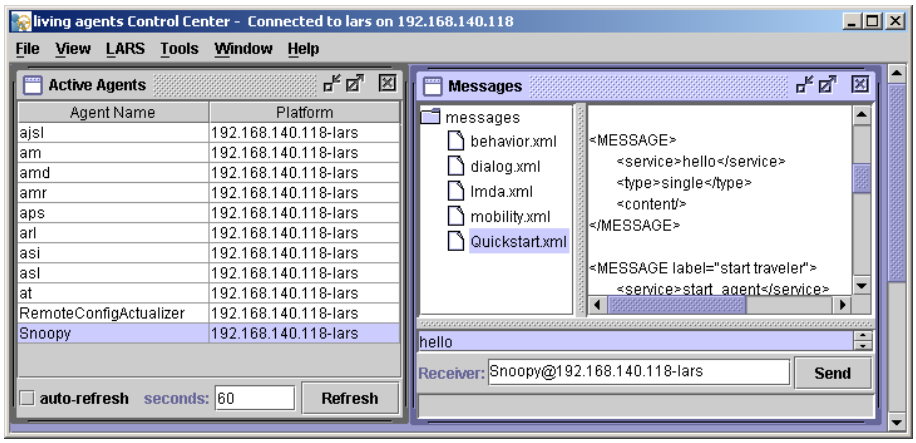

Figure 5.9: LACC agents and messages window to send Snoopy a 'hello' message.

Figure 5.10: Reply of Snoopy to a hello message.

pane of the Woodstock properties window you have to add the 'Swing' and the 'Basics' capabilities, which contains an action that sends a message to another agent. In the services pane you add a service `hi`. Then you add two actions to the `Sequence` branch of the hi service: a `showMessageDialog` action of the `Swing` capabilities to first display a message box saying "Welcome" and a `send` action from the `Basics` capabilities. As parameters for the send action you have to specify the service 'hello' and the receiver 'Snoopy' in the object inspector. The object parameter can be left empty. After saving the settings the agent scenario window will show a link between the two agents indicating that these two agents communicate (Figure 5.11).

5.3.2 Test the Communication

Start Woodstock by pressing the 'Start' button (see Sub-Section 5.2.4). Change to the LACC and press the refresh button on the agents dialog to update the display. Snoopy and Woodstock are displayed in the agent

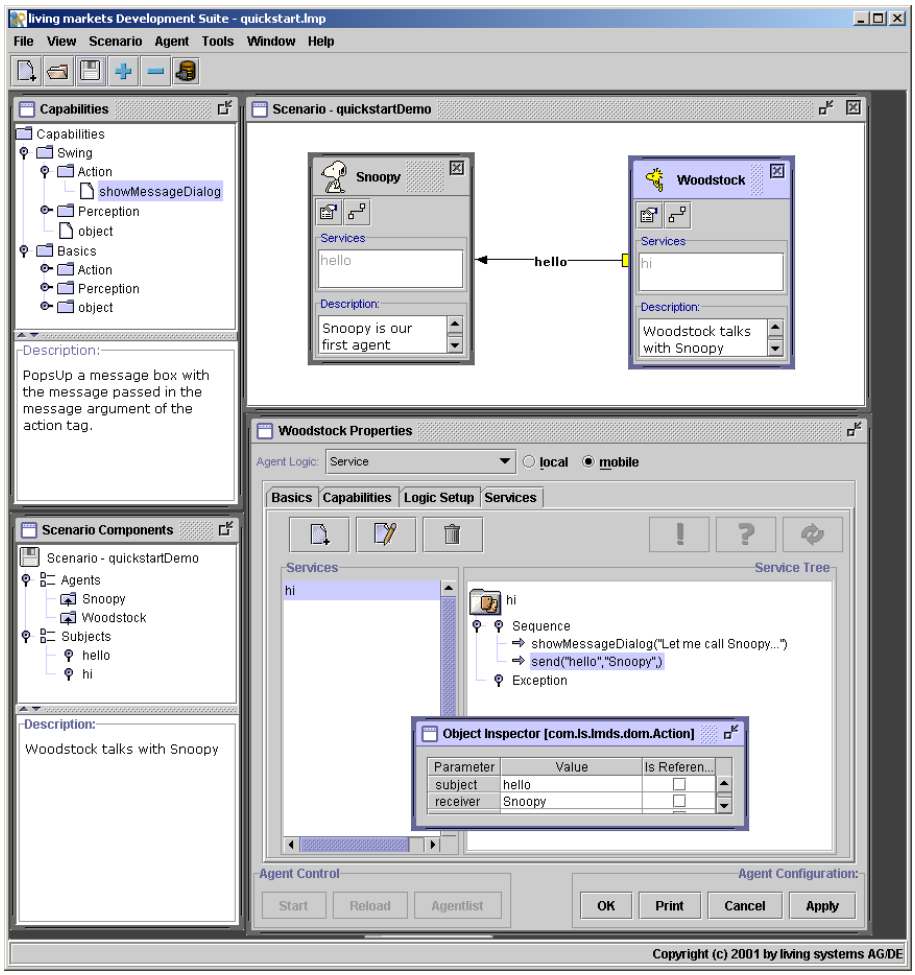

Figure 5.11: LMDS with two agents communicating.

list. Select the `hi` message from the Quickstart message file of the messages dialog. Drag and drop it on Woodstock to send the message to Woodstock. The message box of Woodstocks `hi` service appears with the text "Welcome". After pressing the `OK` button Woodstock sends a `hello` message to Snoopy. Snoopy's message box with the "Hello User!" message appears.

You can now close the agent properties dialog of the LMDS by pressing `OK`.

5.4 Mobility of Agents

One outstanding feature of agents is their ability to move from one computer to another. Instead of sending data to an agent (data-shipping) it could be more efficient to send the agent to the data (code shipping). This applies, for example, if an agent reads tons of data from another server and filters out a small amount. The amount of data sent over the network can be reduced if the agent filters the data on the server where the data resides. In this section we describe the steps necessary to let agents travel around.

5.4.1 Move Agents between two LARSs on one Computer

After installation the system is configured for easily testing mobility of agents between two LARS platforms running on the same computer. The following steps are necessary to move Snoopy from one LARS platform to a second:

1. Start the second LARS platform by using the icon 

. The two LARS platforms will work as a cluster, which is confirmed by a message on the LARS windows:

   ```
   created new [client/server] side messenger
   ```

2. Start a second LACC (see Chapter 4) and manually re-connect it to the LARS2 platform, with menu `LARS>Connect To>lars2`. The list of agents then only shows system agents. You can check the successful clustering of the two platforms by displaying the list of platforms (menu `View>Platforms`). The other LARS should be listed in the list of platforms.

3. Open the list of agents (`View>Agents`) on both LACCs and the messages dialog (`View>Messages`) on the first LACC. Change the auto-refresh rate to 5 seconds on both LACC's agent windows and select auto-refresh.

4. If Snoopy is not already running on the first LARS start Snoopy on the first LARS using the first LACC:

   (a) Select the `Quickstart.xml` file in the messages window.

   (b) Select the agent manager (am) as receiver of the message by clicking on it in the agents window.

(c) Select the `start Snoopy` message in the messages window and press the `Send` button.

5. Migrate Snoopy to the second LARS platform:

   (a) Select Snoopy as receiver of the message by clicking on it in the agents window of the first LACC.

   (b) Select the `migrate to lars2` message in the messages window and press the `Send` button.

   Snoopy disappears on the list of agents of the first LACC and appears on the list of agents on the second LACC[1].

6. Migrate Snoopy back home by sending the `migrate back to home` message to Snoopy on your first LACC.

5.4.2 Move Agents between two Computers

Mobility is much more impressive, when agents move between two LARS platforms running on different computers. Since we do not know your IP addresses in advance, some manual configuration is necessary, before you can move agents between two computers. Since configuration files are only read on startup you have to stop both running LARS platforms. You can now also close the LMDS (make sure you save before exiting) and the two LACCS The following steps are necessary to move agents between two computers:

1. Install **living agents** on a second computer that is connected to the first computer via network. For installation instructions see Chapter 2.

2. Configure both LARS as cluster.

   (a) On both computers you have to change the file

   ```
   $InstDir/conf/lars/lars.cfg
   ```

   Change the part

   ```
   <ipAddress>
       localhost
   </ipAddress>
   ```

   to contain the actual IP addresses of your computers as shown in the example below for my computer:

[1] Since the auto-refresh rate of both agent lists is 5 seconds, it may be displayed on the agent list of the second LACC before the display of the first LACC is refreshed.

```
<ipAddress>
    192.168.140.118
</ipAddress>
```

(b) On your first computer change the file

`$InstDir/conf/agents/lars/PlatformSynchronization.cfg`

The first LARS has to know about the second LARS. This contact information is passed to a LARS agent by a `synchronize_platforms` message. On your first computer you have to specify the connection information that your second LARS prints on startup. Here one example for my first computer:

```
<MESSAGE>
    <service>synchronize_platforms</service>
    <content>
        <platform>
            <platformId>lars</platformId>
            <ipAddress>192.168.50.44</ipAddress>
            <port>8004</port>
            <access>public</access>
            <connectionType>jsocket</connectionType>
        </platform>
    </content>
</MESSAGE>
```

On your computer only the ipAddress should be different.

(c) Also on the first computer change the file

`$InstDir/conf/cockpit/messages/Quickstart.xml`

It specifies the migration messages, for which you have to change the IP addresses of the migration destination. Here the example for my computer:

```
<MESSAGE label="migrate to lars2">
    <service>migrate</service>
    <type>single</type>
    <content>
        192.168.50.44-lars
    </content>
</MESSAGE>

<MESSAGE label="migrate back to home">
    <service>migrate</service>
    <type>single</type>
    <content>
```

```
            192.168.140.118-lars
        </content>
    </MESSAGE>
```

3. Start LARS and LACC on each computer (see Sub-Section 5.4.1).

4. Start Snoopy on your first computer.

5. Test Snoopy by sending it a `hello` message. A message box appears on your first computer. Press the `OK` button.

6. Migrate Snoopy to the second computer (see Sub-Section 5.4.1).

7. Test Snoopy again by sending it a `hello` message from the first computer. A message box appears on your second computer. Press the `OK` button.

8. Migrate Snoopy back to the first computer (see Sub-Section 5.4.1).

9. To check that Snoopy is back again send it a `hello` message. A message box appears on your first computer.

You can now stop LACC and LARS on both computers. For later running the **living markets** Demo Application you should switch back the lars ipAddress of the lars configuration file to localhost:

`$InstDir/conf/lars/lars.cfg`

```
<ipAddress>
    localhost
</ipAddress>
```

5.5 Change/Add Capabilities

In Sub-Section 5.2.3 we described how to create the business logic (behavior) of an agent using capabilities. There are a number of basic and business capabilities already defined within living agents. It is likely, however, that you will want to define your own capabilities. In this section we explain how to create the capabilities, the actions and perceptions that can be used to define the behavior of an agent. For this section you need some Java programming experience since capabilities have to be programmed in Java.

5.5.1 Create a Capability Method

The following steps are necessary to create your own capabilities:

1. Create a class inheriting from class

   `com.ls.logiccontrol.capabilities.LogicCapabilities`

2. For each capability create a method that will be called by the logic engine:

   Actions: Action methods are prefixed by `action` and have the following signature:

   ```
   public void actionTheName(Map logicEnvironment)
   ```

   Perceptions: Perception methods are prefixed by `perception` and have the following signature:

   ```
   public double perceptionTheName(Map logicEnvironment)
   ```

   The return value represents the truth-value of the perception ranging from 0.0 (false) to 1.0 (true).

   The `logicEnvironment` passed to actions and perceptions contains the context of the agent, i.e. the world model and parameters passed to the actions and perceptions. Perceptions may not change the logic environment.

3. For later use in the LMDS, the capabilities have to be documented. All the parameters from the logicEnvironment that are used within an action or perception have to be documented in a javadoc comment ahead of the methods using the `@lsparam` tag.

4. Compile the class and make sure the `build` directory is in your classpath.

5.5.2 Preparation for Use

To use the capability within LMDS you have to provide it with information on the contained actions and perceptions and the parameters they use. This is done by the following steps:

Windows:

1. Open a command prompt by double clicking 

2. Add the `bin` folder of your living agents installation to the `PATH` environment variable:

   ```
   set PATH=%PATH%;$InstDir\bin;
   ```

3. Generate the living markets capability information file (.lmc):

   ```
   ccaps YourClassName.java
   ```

Linux:

1. Open a console window and change to the directory

   ```
   cd $InstDir/source/com/ls/demo/capabilities/;
   ```

2. Generate the living markets capability information file:

   ```
   ccaps.sh YourClassName.java
   ```

Now you can use the capabilities within LMDS. Make sure the `.lmc` file is within the capabilities path of your project properties (see Sub-Section 5.2.1). If your LMDS is already running, you have to reload the environment with menu `Tools>RefreshEnvironment`.

6 living markets Demo Application

The LMDA is designed to demonstrate the application of agents in a logistics environment. Production units produce goods that have to be shipped to consumption units. Transportation units care for shipping the goods and try to optimize their route based on local information they have. They inform other transportation units in their area of slow traffic. All of these units are represented by agents.

This chapter gives an overview on how to use the Demo Application.

6.1 Quickstart Demo Application

Before starting the **living markets** Demo Application, start a LARS server to run the agents created by the Demo Application. To start the LMDA press the icon 

. After startup, the user has two possibilities to proceed. One is to draw a completely new transportation scenario with transportation units, production and consumption units and connections between them. The second is to load a predefined transportation scenario from disk. Here we follow this second path.

A complete transportation scenario is included in the LMDA. It can be loaded by selecting the menu `File>Load Scenario`. The scenario description file is located in the `QuickStart` folder of your installation directory and has the name `QuickStart.scn`. Figure 6.3 shows the transportation scenario and a legend for explanation of the symbols used.

A production unit (see Figure 6.1) produces the products to be shipped. The three bars of a production unit represent the amount of products available,

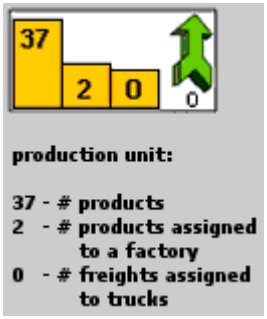

Figure 6.1: Production unit of the LMDA.

Figure 6.2: Consumption unit of the LMDA.

the number of products ordered by a consumption unit and the number of products already assigned to a transportation unit.

A consumption unit (see Figure 6.2) orders and consumes the shipped products. The three bars with numbers of each consumption unit represent the average amount of products that should be available, the number of products ordered from production units and the number of products in the stock.

You start the scenario by pressing the start  button. This will create the agents representing the units of the scenario on the LARS server.

The transportation units will first request for load on production units. This is, as all other communication between agents, indicated by a green dashed line between the agents. Once the transportation units have booked a load they drive to the production unit, load the freight and deliver it to the consumption unit which ordered the freight.

To influence the scenario, the properties of a transportation agent can be changed by right-clicking on the agent and selecting `properties`. In the

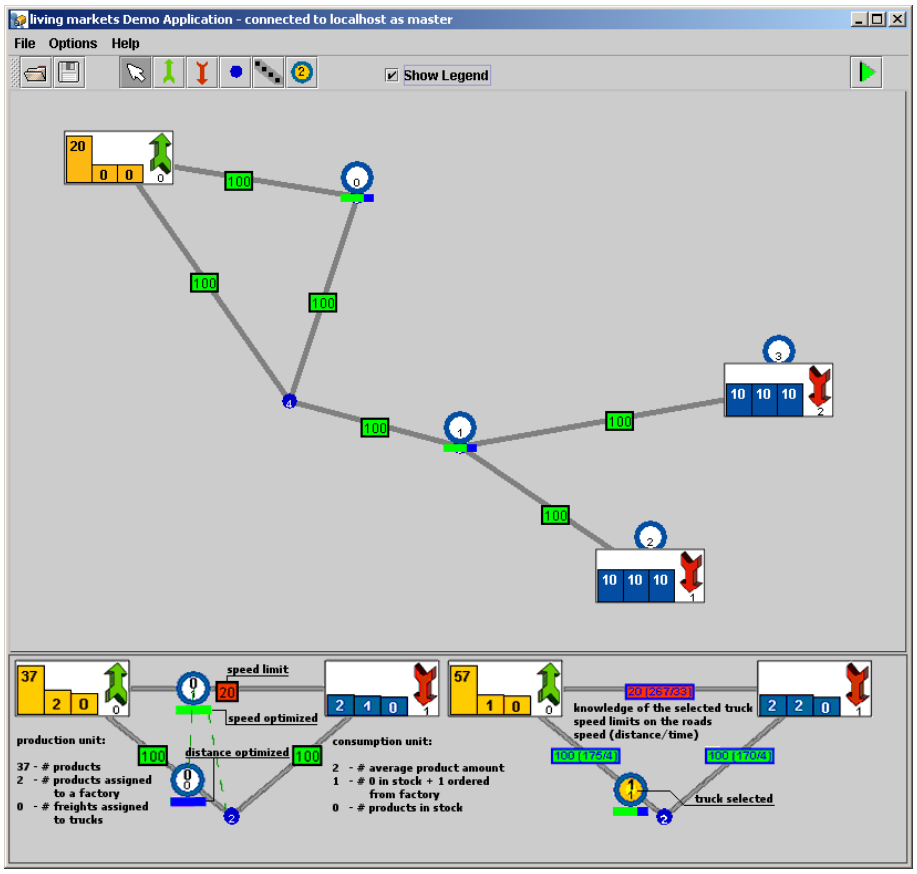

Figure 6.3: A transportation scenario with legend in the **living markets** Demo Application.

agent properties dialog (see Figure 6.4), the trade off of the agent between time and distance (=cost) of travels can be adjusted. A high value of `optimize time` means that the agent will more likely avoid a slow road and prefer a longer road with higher speed.

In the same way the properties of the transportation lines (roads) can be changed by right-clicking on the transportation line and selecting `properties`. In the road properties dialog (see Figure 6.5) the transportation speed can be adjusted to any value between 0 and 100.

By default the road speeds of the simulation are displayed. A click on a transportation agent changes the display to show the road speeds as known by this agent (see Figure 6.6). The agents only know about a slow road if they drove recently on the road or have been informed by other agents. Another click on the agent and the display switches back to true road speeds.

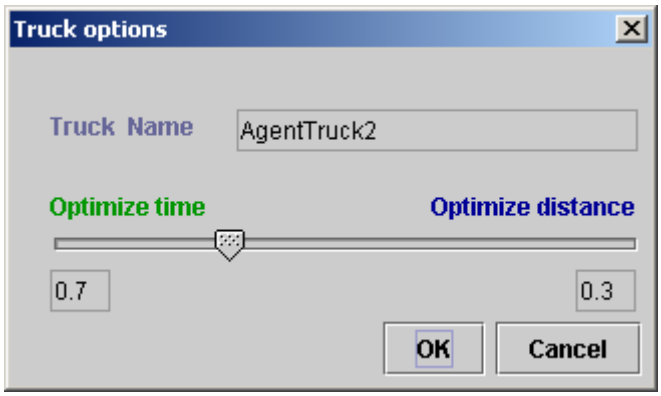

Figure 6.4: The agent properties dialog allows to adjust the trade off of the agent between time and distance of travels.

Finally you can influence the ordering behavior of the consumption units. Right-click on a consumption unit and select `Properties`. The consumption unit dialog (Figure 6.7) allows you to specify the average amount of products that should be on stock.

The LMDA can be stopped by closing the application and the LARS window.

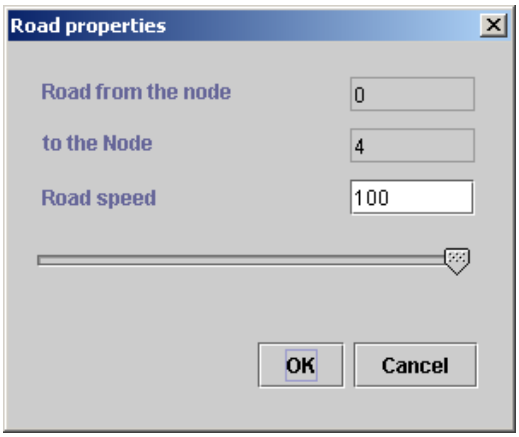

Figure 6.5: The road properties dialog allows to adjust the transportation speed to simulate, for example, traffic jams.

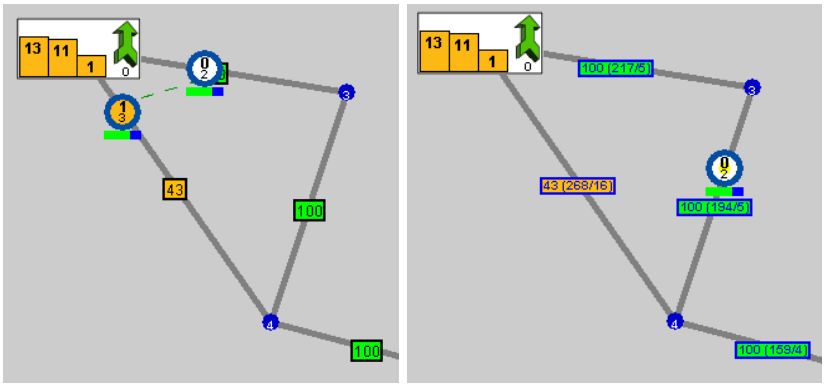

Figure 6.6: Road with reduced speed (left) and information (speed (distance / time)) the agent has about the roads (right).

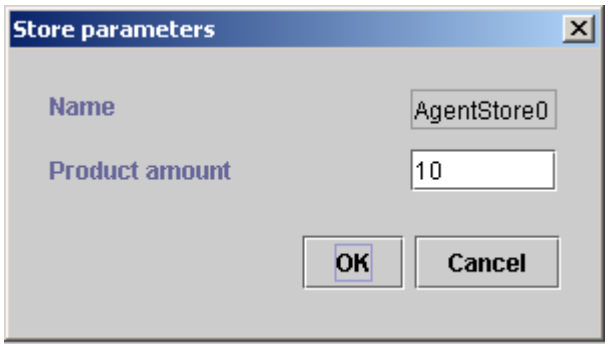

Figure 6.7: Parameters dialog of a consumption unit.

A Troubleshooting

This section helps to solve typical problems while installing and using the living markets product.

Please contact our support team at product.support@living-systems.com, if the following descriptions cannot solve your problem.

Presently there are no problems known.

B Contact Information

B.1 living systems Web Site

You may find more information about **living systems** on our web site at http://www.living-systems.com.

B.2 Technical Support

For technical support please email to product.support@living-systems.com.

B.3 Feedback

Please send all your feedback concerning

- Wish List
- Enhancement Requests
- Bug Reports

to product.feedback@living-systems.com.

B.4 Subsidiaries

All **living systems** subsidiaries are autonomous with marketing and sales responsibilities, including software expertise in customizing our technologies locally and providing local support. All subsidiaries are wholly owned by **living systems** AG.

Global Headquarters:

living systems AG
Humboldtstrasse 11
D-78166 Donaueschingen
Germany

phone: +49 (771) 8987-0
fax: +49 (771) 8987-100

Paris:

living systems France SAS
1, rue La Fontaine
F-77400 Gouvernes
France

phone: +33 (0)1-60 31 50 76
fax: +33 (0)1-60 31 50 76

Singapore:

living systems Asia Pte. Ltd.
8 Temasek Boulevard
#25-02 Suntec Tower Three
Singapore 038988

phone: +65 (887) 5995
fax: +65 (887) 4994

List of Figures

Glossary

EBN Extended Behavior Network

LAC living agents Cockpit

LACC living agents Control Center

LARS living agents Runtime System

LMDA living markets Demo Application

LMDS living markets Development Suite

XML Extended Markup Language