Abstract:
This Deliverable represents the successful production of the software for a small footprint agent shell. This agent shell is called MicroFIPA-OS. MicroFIPA-OS is a lightweight agent platform for small devices such as personal digital assistants. It is based on the component based FIPA-OS agent-programming toolkit. This Deliverable presents an overview of the MicroFIPA-OS system.

MicroFIPA-OS and CRUMPET were demonstrated at the 22\textsuperscript{nd} FIPA meeting in Sendai, Japan [FIP01].

Keyword List:
FIPA, MicroFIPA-OS
Table of Contents

1 INTRODUCTION.................................................................................................................................................................3
   1.1 FEATURES.................................................................................................................................................................3

2 OVERVIEW OF FIPA-OS AND MICROFIPA-OS ....................................................................................................................4
   2.1 FIPA-OS...................................................................................................................................................................4
   2.2 FIPA-OS TRANSPORT STRUCTURE .......................................................................................................................5
   2.3 MICROFIPA-OS ......................................................................................................................................................6

3. CONCLUSIONS..........................................................................................................................................................10

REFERENCES ......................................................................................................................................................................11

APPENDIX A: MICROFIPA-OS USER'S GUIDE .....................................................................................................................12
   INTRODUCTION.....................................................................................................................................................................12
   FEATURES..............................................................................................................................................................................12
   SYSTEM REQUIREMENTS......................................................................................................................................................12
      Setting up the system using the JAR file....................................................................................................................12
   HOW TO BUILD.....................................................................................................................................................................14
      Packages.........................................................................................................................................................................14
      Creating agents..............................................................................................................................................................15
   EXAMPLE AGENTS................................................................................................................................................................15
      Running the first agents on workstations................................................................................................................16
      Example: Communication between two MicroFIPA-OS shells.............................................................................16
      Advanced Configuration..............................................................................................................................................16
   ISSUES.................................................................................................................................................................................17

APPENDIX B: CLASS HIERARCHY OF MICROFIPA-OS ........................................................................................................18
1 Introduction

The goal of Work Package 2 in the CRUMPET project [CRU01] is to develop a small footprint agent shell. This agent shell is called MicroFIPA-OS. The initial design of MicroFIPA-OS can be found in Deliverable 2.1: Design of Small Footprint Agent Platform. MicroFIPA-OS is a lightweight version of the FIPA-OS agent platform [Emo01] for small devices. The implementation is based on the PersonalJava specification [Sun00], supports the PocketPC and Linux operating systems, and supports HTTP as the transport protocol. MicroFIPA-OS and CRUMPET were demonstrated at the 22nd FIPA meeting in Sendai, Japan [FIP01].

The FIPA-OS design and MicroFIPA-OS are presented in Section 1. The class structure and deployment of MicroFIPA-OS agents are discussed in Section 2. Appendix A presents the User’s Guide for MicroFIPA-OS. Appendix B presents the class hierarchy of MicroFIPA-OS.


1.1 Features

MicroFIPA-OS is PersonalJava compatible, it runs on any device that supports the PersonalJava specification and virtual machine and has sufficient memory. MicroFIPA-OS supports FIPA-OS agent programmer’s API and roughly the same functionality as FIPA-OS.

MicroFIPA-OS has been tested on the following devices and operating systems:

- Compaq iPAQ H3600 series (Linux for iPAQ and PocketPC/WindowsCE 3.0&2.11).
- Casio Cassiopeia E115 (PocketPC/WindowsCE 3.0&2.11).
- Psion Series 5mx (Epoc).
- Intel X86 architecture (Linux and Windows95/98/NT/2k).

Supported and tested virtual machines and operating systems are:

- PersonalJava VM by Sun on Linux for iPAQ and PocketPC/WindowsCE 3.0/2.11.
- JDK 1.1.8/1.2.X/1.3.X.
- EpocVM (Psion).

MicroFIPA-OS is able to host a full FIPA platform and execute AMS and DF on the terminal. It has support for FIPA-OS tasks and conversations. Agents that are programmed for FIPA-OS can be driven on MicroFIPA-OS with certain limits. These limits are due to restricted processing power and memory in small devices. MicroFIPA-OS has support for minimal agents that do not use FIPA-OS tasks and conversations. The transport architecture in MicroFIPA-OS is optimised for small devices.
2 Overview of FIPA-OS and MicroFIPA-OS

This section presents the outline of FIPA-OS and MicroFIPA-OS. The differences between the two systems are discussed and the different layers that make up an agent are introduced.

2.1 FIPA-OS

FIPA-OS is an agent development toolkit and a FIPA compatible platform. The FIPA standard [FIP01] aims to improve agent interoperability by providing standards for agent communication language, agent life cycle and how agents interface the platform management and directory functions. FIPA-OS provides the basic high level communication and conversation management functionality, the ability to create tasks and subtasks, the basic platform services specified by FIPA (AMS, DF), and the lower level transport system that supports a set of transports such as RMI and CORBA.

FIPA-OS uses a stack-based layered design, where each agent consists of a number of components that are placed in a stack. The stack is used within the agent, and in addition a transport stack resides within the MTS.

Figure 1 presents an abstract diagram of the FIPA-OS agent stack. The standard FIPA-OS consists of the task and conversation management components, messaging service and the transports. The platform components required by FIPA are implemented as agents, with the exception of the ACC that handles inter-platform communication.

![Figure 1: Abstract architecture behind different versions of FIPA-OS for scaling to different environments.](image)

The standard FIPA-OS agent platform is agent-centric, that is, an agent being executed instantiates MTS based on the profile information (and based on information found in the RMI name service) with its own transport stack. Figure 2 presents an outline of the relationships between different FIPA-OS classes and interfaces.
Figure 2: Relationships between different components in FIPA-OS. The FIPAOSAgent class is the body of the agent. Grey boxes represent interfaces, white boxes instantiable classes.

2.2 FIPA-OS Transport Structure

A class called CommMultiplexService multiplexes a number of different MTPs. If one MTP fails to send a message, the multiplexer tries several other MTPs or sends the message up the stack. Incoming messages are forwarded up the stack; the service handles only outgoing messages.

Figure 3 presents the FIPA-OS service stack, where an agent has an instance of MTS and uses the MessageReceiver interface to receive messages from transports. MTS contains the service stack that contains the internal stack, a number of services and the transports.
2.3 MicroFIPA-OS

MicroFIPA-OS is a scaled down version of FIPA-OS and simplifies the implementation of the FIPA-OS interfaces [LTL01]. Basically, MicroFIPA-OS uses the same interfaces with the exception of the transport mechanism, which has been optimised for the embedded environment. Thus, the MicroFIPA-OS transports are not compatible with FIPA-OS transports without changes. MicroFIPA-OS does not support RMI or CORBA, but by default uses either internal HTTP transport or the FIPA specified HTTP protocol for interoperability.

MicroFIPA-OS agents are programmed in the same way as FIPA-OS agents, using tasks and conversations. MicroFIPA-OS supports a number of agents on the small devices, the sharing of transports and resource pools, and the possibility for local communication. However, tasks, conversations and local messaging present an extra layer of overhead and therefore agent developers may wish to use well-integrated agents and avoid using local messaging.

MicroFIPA-OS agents can be deployed using two different scenarios (figure 4). In the first scenario, MicroFIPA-OS is a part of a greater platform and the FIPA platform agents AMS and DF are running on the fixed network. In the second scenario, MicroFIPA-OS is running an independent agent platform and hosts the AMS and DF. The first scenario necessitates that MicroFIPA-OS and the agent platform are using the same internal transport (which can also be a FIPA specified protocol) and that the agent platform is able to forward messages to the mobile node. The latter scenario requires that MicroFIPA-OS is running a FIPA compatible transport and able to receive messages and forward them to the proper agents.

Currently, MicroFIPA-OS has a lightweight ACC (Agent Communication Channel) that forwards incoming messages based on their recipient agent identifiers.
One of the goals of the implementation presented in this document was to modify the FIPA-OS to a more platform-centric idea that there may be more than one agent in a single VM with one instance of MTS and one set of transport protocols. We only need a single transport stack for all the agents. On the other hand, in order to keep compatibility with FIPA-OS we needed to use the same interface definitions.
MicroFIPA-OS supports different configurations using the FIPA-OS interfaces. Basically, components such as the ConversationManager, TaskManager and MTS may be changed for different implementations. Figure 5 presents a possible configuration for MicroFIPA-OS following the FIPA-OS layer based structure.

![Figure 5: Two concrete realisations using pluggable components. The left stack represents a traditional FIPA-OS agent and the right stack corresponds to a scaled down shell for the agent.](image)

MicroFIPA-OS uses the FIPAOSAgentComponentFactory-interface for configuring the various components. This interface defines the TaskManager, ConversationManager and MTS components that are to be used by the agent. MicroFIPA-OS supports two ways of creating agents. The first way is the traditional FIPA-OS way of programming using tasks and conversations. Agents such as the PingAgent can be executed on the small device. However, if the device is limited it may not be reasonable to run very heavy infrastructure to support agents. Therefore, MicroFIPA-OS also supports a more minimalistic way of creating agents. This minimal mode is also internally configured using the AgentComponentFactory interface. Minimal agents use the same parsers and messaging features of the MicroFIPA-OS, including transports, but they do not use conversations or tasks. Messages are directly delivered to the corresponding callback methods and agent programmers have to implement the necessary behaviour. This enables the programmer to create thinner agents.
The following methods are provided for minimal agents:

- For sending messages:
  
  o public void send(Message msg)

- For handling incoming messages from other agents: public void
  
  o handleMessage(Message msg)

- For handling errors pertaining to messages. A Typical error is that
  
  the message cannot be delivered.

- public void handleError(Message msg)
3. Conclusions

MicroFIPA-OS is an efficient agent platform for small devices. It is PersonalJava compatible and it can be run on any device that supports the PersonalJava specification and virtual machine and has sufficient memory. MicroFIPA-OS supports FIPA-OS agent programmer’s API.

MicroFIPA-OS can provide a full FIPA compliant agent platform with AMS (Agent Management System Agent) and DF (Directory Facilitator Agent). Another option is to run MicroFIPA-OS as part of another platform or as an aggregate of another platform. Thus limited memory and performance resources can be saved.

References


Appendix A: MicroFIPA-OS User's Guide

Introduction

This document provides instructions on how to setup and configure the MicroFIPA-OS for Linux and Windows. MicroFIPA-OS is a small-footprint version of the open-source FIPA-OS that provides the FIPA-OS API compatible "shell" that is executable on small devices.

This release is aimed for developing and testing MicroFIPA-OS agents and may contain features that are not needed on the small device, such as AMS and DF functionality and, for example, the SL and SL0 parser.

Features

This MicroFIPA-OS release includes the following functionality:

- Task and Conversation management
- Support for the following FIPA specified interaction protocols at the conversation manager level: fipa-request, fipa-query family, fipa-subscribe.
- Support for AMS and DF registration, deregistration and query using the FIPA-OS API.
- Support for the SL content language.
- Lightweight XML parser (MicroStar Aelfred).
- FIPA-OS profile support encapsulated in the MicroFIPAOS - class.
- Support for multiple agents on the terminal.
- Support for http internal protocol, and optionally FIPA compatible http protocol.

System requirements

Java Virtual Machine (JVM): JDK 1.1.8+ or Kaffe 1.04 or 1.06 (Intel and ARM). It has also been tested with the PocketPC ARM PersonalJava.

The following JAR files in the CLASSPATH (included in the release):


Setting up the System using the JAR file

This section describes how to run the first agent using the binary version of MicroFIPA-OS included in the JAR-file.
Linux

3 Create a suitable directory to install the software under your home directory. For example "mfos" in "userhome/mfos/".

4 Copy the MicroFIPA-OS zip-file "mfos_vx.x.zip" into that directory and unzip it. This will create the subdirectories bin, src and doc. The bin-directory contains jar-files and the src-directory contains the source-code.

5 Now you need to define the proper classpaths. This can be done in two ways:

5.1 Define the classpath on the command line:

First navigate to the directory "userhome/mfos/" Then type "export
CLASSPATH=userhome/mfos/bin/mfos.jar:userhome/mfos/bin/collections.jar:userhome/mfos/bin/sax.jar:userhome/mfos/bin/aelfred.jar:userhome/mfos/bin/SiRPAC-1.14.jar" You can also leave the path out, but then the jar-files are related to the current directory; if you are in directory "mfos/" and want to use short paths then the command should be "export
CLASSPATH=bin/mfos.jar:bin/collections.jar:bin/sax.jar:bin/aelfred.jar:bin/SiRPAC-1.14.jar:.
"

5.2 Edit the file ".profile":

The classpath can also be defined in a file called ".profile" that should be situated under your home directory. There you can specify paths pointing to the same jar-files as in the above example. However, in ".profile" the paths are to be perfect, like "system/userhome/mfos/jar".

The command should be in form: export CLASSPATH ="/system/userhome/mfos/bin/mfos.jar:/system/userhome/mfos/bin/sax.jar:/system/userhome/mfos/bin/aelfred.jar:/system/userhome/mfos/bin/collections.jar:/system/userhome/mfos/bin/SiRPAC-1.14.jar:"

Exception in using the ".profile":

Let's assume that you've just set the CLASSPATH to ".profile" via a Linux terminal. The CLASSPATH is not yet functional in that terminal, or in other terminals that have been opened before the setting. To make the settings in ".profile" functional you have to open a terminal after the setting has been done.

6 Type "java microfipaos.test.LocalTest localhost:10000" or "kaffe microfipaos.test.LocalTest localhost:10000". The example program has been tested with JDK 1.1.8 and Kaffe, so those versions are recommended. This command uses the mfos.jar file. If you use Kaffe, see Section 7.

7 You have just run your first MicroFIPA-OS agent. Try more example-agents that are described in Section 5.
Windows Console

- First create a directory for the MicroFIPAOS. Note that the java environment should be operable in this directory i.e. you should be able to execute java and javac commands in this directory. Let us call the directory "C:\mfos"

- Copy the file MicroFIPA-OS zip-file into the "C:\mfos"-directory and unzip the file. This will create two subdirectories; fipaos and microfipaos. This will create the subdirectories bin, src and doc. The bin-directory contains the jar-files and the "src"-directory contains the source-code.

- Type "set CLASSPATH=c:\mfos\bin\mfos.jar; c:\mfos\bin\collections.jar; c:\mfos\bin\sax.jar; c:\mfos\bin\aelfred.jar;." you may also omit the full path, but then the classpath is relative to the current directory. Note that the separator of the files included in the CLASSPATH is ";" in Windows, not ":" like in Linux.

- Type "java microfipaos.test.LocalTest localhost:10000" or "kaffe microfipaos.test.LocalTest localhost:10000"

- You have just run your first MicroFIPA-OS agent.

How to build

This release includes the full source code to the MicroFIPA-OS, which you can build using either the standard Java command line compiler (javac) or, for example, some graphical Java development tool such as JCreator.

You need to have the following JAR files in your CLASSPATH in order to compile: collections.jar, sax.jar, aelfred.jar and SiRPAC-1.14.jar. You should not have mfos.jar in your CLASSPATH when you are compiling. In addition, the root of the packages (where the fipaos and microfipaos directories are located, by default the src directory) needs to be in the CLASSPATH.

For example, if you are situated in the directory "userhome/mfos/" then type "export CLASSPATH=bin/src:/bin/collections.jar:bin/sax.jar:bin/aelfred.jar:bin/SiRPAC-1.14.jar:".

After the classpath is configured files are compiled from the root (in our example userhome/mfos/src) using the standard javac:

In Linux

"javac microfipaos/agent/MicroFIPAOSAgent.java"

In Windows:

"javac microfipaos\agent\MicroFIPAOSAgent.java"

PACKAGES

The MicroFIPA-OS distribution consists of two packages:
fipaos | Standard FIPAOS classes, including SL and XML parsers, AMS and DF, and database functionality.
---|---
microfipaos | MicroFIPA-OS classes and example agents.

**CREATING AGENTS**

MicroFIPA-OS agents are created using the standard FIPA-OS programming API. In order to create a more efficient code, the handling of task callback events should be done in a special dispatcher method. The example agents, for example SearchTest, demonstrate how to use the callbacks.

**Example agents**
The following example agents are included in this distribution (examples are in the microfipaos.test package):

<table>
<thead>
<tr>
<th>Agent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LocalTest</td>
<td>This agent starts AMS and two other agents locally. One agent sends messages to the other agent periodically using WaitTask. LocalTest displays the time spent in passing one message between the agents in milliseconds.</td>
</tr>
<tr>
<td>LocalTest2</td>
<td>Same as above but AMS is not used.</td>
</tr>
<tr>
<td>StartAMSDF</td>
<td>Starts AMS and DF plus one test agent. Takes the address as the command line parameter: for example StartAMSDF localhost:12000 starts the system on the current host on port 12000. StartAMSDF prints the time in milliseconds between incoming messages for the test agent.</td>
</tr>
<tr>
<td>Tester</td>
<td>This agent sends periodical messages to the test agent started with StartAMSDF. The first command line parameter determines the location of the other agent (Access node) and the second parameter is the local address. For example: Tester localhost:12000 localhost:14000 makes the agent send messages to port 12000 and receive messages on port 14000.</td>
</tr>
<tr>
<td>SearchTest</td>
<td>This agent registers to AMS and DF and</td>
</tr>
</tbody>
</table>
searches the DF for an entry. The command line parameters are similar to the Tester agent.

**RUNNING THE FIRST AGENTS ON WORKSTATIONS**

- **LocalTest**: LocalTest is executed by first setting the correct `CLASSPATH` (if it is not already set) and then using the command: 
  
  ```
  java microfipaos.test.LocalTest
  ```
  
  This runs the internal transport on port 10000 on the current machine and starts the agent.

- **StartAMSDF**: This agent is executed in the same way as LocalTest, but requires a parameter, for example 
  
  ```
  StartAMSDF localhost:10000
  ```
  
  After AMS, DF and the test agent have been initiated the test agent waits for interaction.

- **Tester**: The tester first requires the address of the access node and then its own address, for example: 
  
  ```
  java microfipaos.test.Tester localhost:10000 localhost:12000
  ```
  
  Now, the agent tries to contact the AMS at port 12000 and accepts messages on port 10000. Tester and StartAMSDF start to communicate.

- **SearchTest**: SearchTest is similar to Tester and outputs the results from the DF search.

**EXAMPLE: COMMUNICATION BETWEEN TWO MICROFIPA-OS SHELLS**

To demonstrate message passing between two MicroFIPA-OS shells running either on the same machine or on different machines, first startup the StartAMSDF agent on a computer and configure it using that computer's IP-address and ports. Then setup the Tester agent on another port or some other computer and give the AMSDF agent address including the port as the first parameter, as the second parameter give the local IP and desired port.

**ADVANCED CONFIGURATION**

The static class `microfipaos.agent.MicroFIPAOS` holds the configuration information of the shell, and it may be modified when starting agents (for example in the constructor) to change the internal address, the access node address and some other parameters. Here is a short list of the parameters that affect the execution of the shell:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Internal</strong></td>
<td>Default internal address (localhost:10000)</td>
</tr>
<tr>
<td><strong>Accessnode</strong></td>
<td>Default address of the access node. (localhost:10000)</td>
</tr>
<tr>
<td><strong>DEFAULT_HAP_NAME</strong></td>
<td>Default local agent platform name</td>
</tr>
</tbody>
</table>
DEFAULT_TM_LOCAL_THREADS | Number of local threads in the thread pool.
---|---
DEFAULT_DB | The default database name, for example "MemoryDatabase" or "NoDatabase"
ConversationPoolSize | Conversation pool size
allowConversationCloning | Allow the cloning of conversations. Improves concurrency with the expense of increase memory consumption. Default is true.

**Bug Reporting**

A list of discovered and solved bugs is available at http://www.cs.helsinki.fi/group/crumpet/mfos/. Here you can also download the latest version of the MicroFIPA-OS that might include the bug fix to your problem. If you find an unreported bug, please, report it to crumpet@cs.helsinki.fi. Include as much information as you can in the bug report. Especially debug messages and stack traces with line numbers are useful, and also the MicroFIPA-OS version number, Java environment, operating system, date of discovery, description about the bug and example code that causes the error.

**Issues**

- If a socket exception is thrown, the reason is probably a wrong port number, which may be already in use by some other program.

- Kaffe does not properly support the translation of localhost to IP, so the IP address of the host is required as a parameter. If the normal command-line parameter was in string-form: "localhost:portnumber", then it should be in numerical-form: "128.214.9.199:portnumber". In Linux you can get the IP address of your computer with the command "host String-form address" where the address in String-form is like: "MachineName.cs.helsinki.fi". To check the correctness of the numerical-form IP address you can use "host numerical-form" that returns the host name.

- The agree message may arrive before the inform message, which may result in some cases where the task is being terminated upon the inform message and no task is listening for the agree message. This may lead to a not-understood message being sent back to the sending party.

- MicroFIPA-OS debug messages can be activated for most of the classes (such as MTSLite.java) by using a boolean debug variable. When the debug is turned on, the system prints debug messages to the console.

- If the default diagnostics is turned off. The diagnostics class contains two classes DIAGNOSTICS and DIAGNOSTICSL. You can reactivate the diagnostics output from the code by changing the DIAGNOSTICSL class within the Java file in the fipaos.util-package to its original name DIAGNOSTICS and making it public. You must change the original class to some other name and make it private.
Appendix B: Class Hierarchy of MicroFIPA-OS

- class java.lang.Object
  - class fipaos.agent.FIPAOSAgent (implements fipaos.agent.conversation.ConversationListener)
    - class fipaos.platform.AgentManagementSystem
      - class microfipaos.platform.AgentManagementSystem
    - class fipaos.platform.DirectoryFacilitator
      - class microfipaos.platform.DirectoryFacilitator
    - class microfipaos.agent.MicroFIPAOSAgent (implements fipaos.mts.MessageReceiver)
      - class microfipaos.test.LocalTest
      - class microfipaos.test.LocalTest2
      - class microfipaos.test.LTester
      - class microfipaos.test.SearchTest
      - class microfipaos.testTester
  - class microfipaos.agent.MicroFIPAOS
  - class microfipaos.agent.MicroFIPAOSAgentComponentFactory (implements fipaos.agent.FIPAOSAgentComponentFactory)
  - class fipaos.mts.MTS (implements fipaos.mts.MessageSender)
    - class microfipaos.mts.MTLSlite
  - class microfipaos.mts.Multiplex (implements fipaos.mts.http.ICallback)
  - class microfipaos.test.StartAMSDF
  - class fipaos.agent.task.Task
    - class microfipaos.test.LocalTest.IdleTaskTest
    - class microfipaos.test.LocalTest2.IdleTaskTest2
    - class microfipaos.test.SearchTest.IdleSearchTask
    - class microfipaos.testTester.IdleTask2