Facilitating Agent Messaging on PDAs

Sasu Tarkoma [1] and Mikko Laukkanen [2]

[1] Helsinki Institute for Information Technology (HIIT) and
Department of Computer Science
P.O.Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki
+358 9 191 44263
sasu.tarkoma@{hiit.fi, cs.helsinki.fi}
[2] Sonera Corporation
P.O.Box 970 (Teollisuuskatu 13)
FIN-00051 SONERA
+358 40 5073358
mikko.t.laukkanen@sonera.com

Abstract. Java and middleware are becoming popular on PDAs and small devices with the advent of small-footprint virtual machines. In this paper, we examine agent messaging on small devices based on our experiences with MicroFIPA-OS, which is an agent toolkit for small devices. Since agents communicate by sending and receiving messages, it is reasonable to expect that the system should minimize communication latency both within a platform and between agent platforms. We present and discuss different deployment options for connecting terminal agents with the fixed network agent community. The performance measurement focuses on messaging both within the PDA device and between the device and another external agent platform.

1 Introduction

Personal Digital Assistants (PDAs), smart phones and various wireless and wireline communication mechanisms enable ubiquitous and nomadic computing. Currently Java is being introduced into mobile devices in the form of small-footprint virtual machines. Almost all PDA devices support Java at least to some degree, and also many of the current and future mobile phones are supporting Java either in the form of applets or midlets. This progress in hardware and software has made it possible to deploy high-level processes on small devices, for example software agents [7].

Agents rely on asynchronous message delivery, which should be efficient and reliable [9]. Since agents communicate by sending and receiving messages, it is reasonable to expect that the system should minimize communication latency both within a platform and between agent platforms. Message based communication is well suited to ubiquitous environments, because it is asynchronous in nature and message queuing makes it easier to support disconnected operation.

Because of the large variety in the capabilities of the devices, the agent execution environment must be scalable to allow different configurations for different environments. We present our experiences with MicroFIPA-OS, which is derived from the FIPA-OS toolkit [13] that is based on the FIPA specifications. Both MicroFIPA-OS and FIPA-OS are available as Open Source from the SourceForge-forum [16]. FIPA[1] is a standardization organization that promotes interoperability between agent platforms [3]. MicroFIPA-OS and its core components can be distributed between several devices. Distribution supports the execution of computationally heavy components on more powerful hosts. In this paper, we focus on agent messaging within a single terminal and in a partial platform environment, where a single device supports the execution of one or more agents that are part of a greater domain.

We examine the performance of the MicroFIPA-OS system on two current PDA devices, Compaq iPAQ [2] and Casio Cassiopeia E-115. MicroFIPA-OS was developed in the CRUMPET project [12] by the University of Helsinki. CRUMPET aims to provide tourism services for mobile users. The system is used as the client side agent execution environment. The platform allows the client to access agent based tourism services and services mediated by agents, and to communicate information pertaining to user profile, location and communication link to the service agent community.

This paper is structured as follows: the rest of this section introduces the FIPA architecture, Java on small devices and related work. Section 2 presents an overview of MicroFIPA-OS and examines agent deployment on small devices. Section 3 presents the performance tests that measure the effiency of both local and external communication on the small device. Finally, Section 4 presents the conclusions.

1.1 FIPA Architecture

The FIPA architecture [3] aims to improve agent interoperability by providing standards for message transport protocols, message envelopes, agent communication language (ACL), content languages and interaction protocols. Message transport protocol specifies how the agent messages are transferred. The Envelope layer allows message transport protocol-independent message handling. The ACL layer defines the language and structure of messages used in agent communication, and the content language describes the content of the messages. On the highest level the interaction protocol defines the interaction pattern, which determines how agents exchange messages. These layers can be structured as a communication stack, which can be seen in Figure 1.

In addition to the communication stack, FIPA defines two central components of a FIPA agent platform, the Agent Management System (AMS) and the Directory Facilitator (DF) agents. AMS manages the life cycles of the agents residing on the platform and DF provides agents with yellow page services, i.e. allows agents to advertise their capabilities and to search for agents on the basis of their advertised capabilities. MicroFIPA-OS can be run in a partial-mode, where the AMS and/or DF can be distributed between two hosts.

[1] Foundation for Intelligent Physical Agents, see http://www.fipa.org.

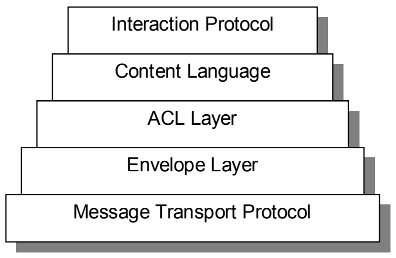

Fig. 1. FIPA communication stack

1.2 Java on Small Devices

The benefits of Java are portability of code and becoming less attached to the operating system and hardware. The choice of the Java specification affects performance, memory footprint, portability and reuse of code. The older PersonalJava-specification intended for PDA devices has both open-source and commercial implementations available [17]. The Java 2 Micro Edition (J2ME) consists of several configurations targeted at PDA-devices and mobile phones. The Connected Device Configuration (CDC) is aimed at PDAs and is complemented by different profiles. PersonalJava is supported in J2ME through the use of CDC and PersonalProfile. Moreover, the Connected Limited Device Configuration (CLDC) has been designed for lower-end devices such as smart phones [18].

CLDC was found to be too limiting for the requirements of the FIPA-OS API and since the PersonalProfile for CDC was not available at the time, PersonalJava was chosen as the Java programming environment. In addition to PersonalJava and proprietary lightweight Java implementations, it is also possible to run MicroFIPA-OS on standard editions of Java (Java 1.1 and J2SE) and on CDC.

1.3 Related Work

The performance measurements presented in this paper are based on sockets-based communication using a proprietary protocol. However, there are several Java-based messaging solutions for embedded and wireless use. The Java Messaging Service (JMS) is a messaging interface specification that supports both point-to-point and publish/subscribe model communication. JMS is also being used on small devices as a messaging solution, for example the iBus//Mobile, which uses a gateway [15]. JMS is one option that can be used as the basis for agent communication on small devices; however, solutions for the problematic last hop are required.

Several small-footprint CORBA ORBs [10] are also available for different operating systems and programming languages. For example, Orbix/E 2.0 from IONA [6]. However, in general RPC-style communication is synchronous, and not suitable for wireless communication subject to disconnections and long round-trip times. This motivates the use of message-oriented middleware.

Related work in executing software agents on small devices has been done in the LEAP project, which is developing a lightweight agent system derived from the JADE agent platform [1]. The system is targeted for lower end devices and the CLDC Java configuration.

2 MicroFIPA-OS

MicroFIPA-OS is an agent development toolkit and platform based on the FIPA-OS agent toolkit. The system targets medium to high-end PDA devices that have sufficient resources to execute a PersonalJava compatible virtual machine. MicroFIPA-OS simplifies the FIPA-OS agent internals, does not mandate XML-parsing nor the use of XML profiles, introduces thread and resource pools to reduce the overhead in frequent object creation, and reduces the number of dynamic method invocations. MicroFIPA-OS allows the use of FIPA-OS components such as AMS and DF.

FIPA-OS and MicroFIPA-OS use tasks and conversations as the basic metaphor for programming agents, and agent functionality. Since the task and conversation management introduce overhead and latencies in messaging and agent execution, MicroFIPA-OS also supports a minimal mode, which has a minimal API for sending and receiving messages. Agent developers, who require better performance and guarantees, can implement their own management components using this mode.

2.1 Messaging

MicroFIPA-OS supports the same communication stack as FIPA-OS; however, the message transport layer and the interaction protocol layer are simplified for operation in resource-constrained environments. Furthermore, the message transport protocol layer, the envelope layer and the ACL layer can be enhanced by plugging in nomadic application support, which provides message transport protocol especially designed for wireless communication [9], and bit-efficient encodings for the envelopes and ACL messages [5].

FIPA-OS relies on Java RMI for internal communication and the centralized ACC for external communication. In a typical configuration, agents and the ACC reside in different virtual machines, which is not reasonable on small devices. Moreover, the system allocates an internal transport stack for each agent. These issues were addressed in MicroFIPA-OS by redesigning the transport structure and introducing a component called the Multiplexer, which maintains the set of transport protocols for all agents. MicroFIPA-OS currently supports a sockets-based internal transport and a HTTP-based external transport for interoperability. The system can also be configured to use other transports. The transports may implement their own buffering and delivery policy. If a transport cannot deliver a message it will be returned to the sending agent and marked as undeliverable. In addition, with the Multiplexer it is possible to try several different transport mechanisms if there are transmission failures

2.2 Deployment

Currently, most agent platforms are stationary, long-lived and support many agents. This is in contrast with the ubiquitous environment created by the current and forthcoming small devices, which support the execution of software agents. This creates a number of different configurations typically using either the peer-to-peer model or the client-server model.

In peer-to-peer operation, the whole platform is potentially mobile and interoperates with other platforms using standardized protocols. The client-server configurations have either browser-clients with limited support for application logic or more complex client-side software modules. This includes client systems that use a proxy and support only a small amount of logic, usually the user interface, on the client system [14]. The clients are connected with the agent world using various protocols, usually HTTP, and access services provided by the server-side agent system using a proxy or a gateway [6,9]. If most of the functionality is implemented at the server-side, the system becomes inoperable when disconnected and may suffer from the long round-trip times of wireless links.

If the client-side system is capable of executing agents and supports the agent communication language the client may use the server-side system more freely without relying on the communication patterns programmed in the proxy. This kind of partial platform operation bridges the agents on the small device with the fixed network agent world. The benefits of this approach is that agent code can be potentially reused, agent communication language is used throughout the domain, and the possibility of executing client-side logic improves response times when network access is not necessary or feasible, and may enable partial functionality when the network is inaccessible. This approach requires, however, more system resources than the proxy-based solutions and requires solutions for the management of platform parts and may require the use of gateways.

The minimum level of support for agent messaging is the agent communication language, which is the lingua franca of agent systems. Without supporting ACL the agents on the small device are proprietary software interacting with the agent system using intermediaries. A better support for agents on small devices is to incorporate the notion of tasks or behaviors, concurrent operation, and conversations, which ease the management of high-level interaction protocols between an agent and its peers.

MicroFIPA-OS supports these two models for creating and running agents. In the first mode application programmers that use the FIPA-OS API may reuse their existing code developed for the FIPA-OS platform. In the second mode, developers use ACL, envelopes, and content language parsers provided by the toolkit, but write their own message and task handling code. The general usage scenario for MicroFIPA-OS is the partial platform scenario, where agents use the AMS and DF of a long-lived and stabile fixed network platform. However, the system also supports the possibility of running an independent system on the small device, including AMS and DF, and interfacing other FIPA systems with the HTTP transport.

2.3 Nomadic Environments

The architecture of MicroFIPA-OS is extensible by plugging in components that either replace or extend the architecture. An example of this kind of contribution is Nomadic Application Support (NAS), which provides support for wireless environments. The main contributions in NAS are the agent naming management, the wireless message transport protocol [9] and bit-efficient encodings for the envelopes and ACL messages [5].

MicroFIPA-OS does not provide mechanisms for resolving agent naming issues. It is the task of the AMS to enforce naming policies within a platform. NAS solves this problem by introducing a concept of terminal ID, which is a unique identifier for a device. The terminal ID is an addition to the agent name and the use of terminal ID is required in order to prevent agents with the same name running on different devices that are part of the same domain. In addition, by using the persistent terminal ID, the possible network address changes can be handled seamlessly.

In some wireless and firewall scenarios it is not possible to push data to the terminal. For example, server-side push is not possible in MIDP [18] devices that support only HTTP. Moreover, if the mobile device is behind a NAT-box (Network Address Translation), it is typically not publicly addressable. This disables the possibility to push messages (i.e. open connections) to the mobile device and the only way to allow bi-directional messaging is to let the mobile device open a persistent connection. NAS introduces a wireless message transport protocol that implements this approach. For more information about the design and performance, see [9]. NAS allows MicroFIPA-OS to support bit-efficient envelopes and ACL messaging [5]. The bit-efficient codec can be selected at runtime and on message-basis.

3 Internal and External Performance

This section examines the hardware environment and the MicroFIPA-OS system performance on a Linux laptop and two current PDA devices: the Compaq iPAQ and Casio Cassiopeia E-115. We focus on the two deployment scenarios; first the performance of FIPA messaging when AMS and DF are local to the PDA, and the case where AMS and DF are located on some other host.

3.1 The Environment

The small device environment poses several challenges that need to be met: limited amount of RAM memory and flash memory, limited CPU performance, battery life and connectivity [8]. The performance results presented in this section were achieved using the configurations illustrated in Table 1. MicroFIPA-OS was executed without a GUI and the diagnostics were turned off for the tests. The CaffeineMark benchmark [11] is included in the table and represents the general virtual machine performance of the different configurations.

Table 1. Device configuration

Device	**Casio Cassiopeia E-115**	**Compaq iPAQ H3630**	**Laptop**
Operating system	Windows CE 2.11	Linux 2.4.3	Linux 2.2.16
RAM	32MB	32MB	192MB
Storage	16MB	16MB	6GB
JVM	Sun PJava 1.0	Sun JDK1.1.8	Sun JDK 1.1.8
CaffeineMark	27	120	679

3.2 Local Performance

Table 2 presents the measured performance of local messaging using DF search and FIPA-request messaging [4]. In DF-search the system forms a DF query object, which is passed to the DF agent. The DF agent performs the search with the given constraints and returns a result object back to the agent. We can see that the DF search operation is computationally too heavy for a low-end device.

Table 2. Device configuration and local performance. The approximate runtime footprint includes the virtual machine overhead, and the FIPA-request test agent. The request message is a simple ping

	Casio Cassiopeia E-115	**Compaq iPAQ H3630**	**Laptop**
Startup (ms) ams+df+test agent	16574.5 +/- 1509	2206.5 +/- 1.58	439.2 +/- 32.1
DF-search (ms)	7765.6 +/- 568	932.8 +/- 211	153.51 +/- 11
FIPA-request (ms)	861.3 +/- 484.2	144 +/- 22	18.1 +/- 9.6
FIPA-request minimal (ms)	161.2 +/- 15.7	37.3 +/- 0.48	10.2 +/- 0.42
Runtime footprint	3 MB	4.1MB	3.9 MB

In FIPA-request the protocol initiator sends a request-message, and the recipient responds first with an agree-message and then with an inform-message. All the FIPA platform agents use this protocol for registration, making modifications and searching white and yellow page directories. The case of minimal API discussed in Section 2 is presented for comparison, and it performs better than the regular API that is compatible with FIPA-OS. The difference between the minimal and regular API stems from the presence of the task management and conversation management layers in the regular API. Both APIs use the same message transport mechanism. Results show that when using a low-end device, the use of minimal mode is well justified in terms of performance.

3.3 External Performance

External performance was measured by executing the test agent on the small device and the AMS and DF were executed on a laptop. Figure 2 presents the results of DF-search and FIPA-request messaging over 115200, 57600 and 9600 bits/s data communication link (RS232C) using the sockets-based internal transport of MicroFIPA-OS. For comparison the ping utility values with 108-byte payload are 30.4 ms for 115Kbps, 60.28 ms for 57.6Kbs and 296.7 ms for 9.6Kbps. The y-axis depicts the total time required by the interaction. DF-search uses FIPA-request and includes the DF agent processing overhead at the fixed network. Since the internal transport does not use serialization to transfer the objects they need to be parsed either from ASCII or bit-efficient representation. The tests in this paper use ASCII representation. The communication link was realized using standard cabling without data compression. The results do not portray wireless transmission (although 9600 bps represents the basic throughput of GSM data), because the high variability and long delays in a live wireless network are not modeled.

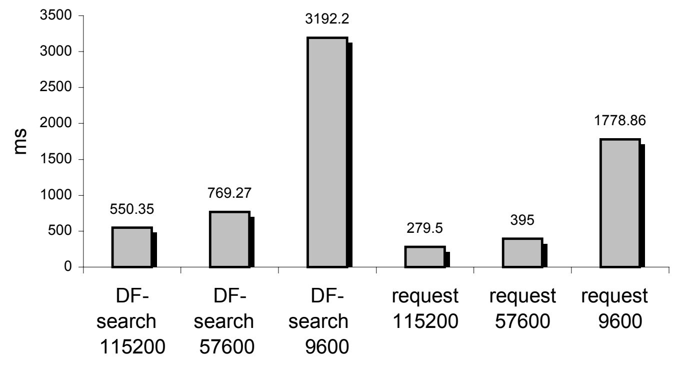

Fig. 2. Average DF-Search and FIPA-request performance in milliseconds using data communication on the Compaq iPAQ with Linux

3.4 Discussion

The benchmarks indicate that the task and conversation management in MicroFIPA-OS, which are compliant with their FIPA-OS counterparts, introduce overhead. Compared with our previous experiences with FIPA-OS 1.3.0 on Casio E-115, local FIPA-request is more than 23 times faster, and DF-search is approximately 4 times faster on MicroFIPA-OS [8]. For low-end devices these increases in performance may not be enough for complex functionality. The overhead can be avoided by using the minimal API, which performs better, but on the other hand requires that agent programmers introduce their own task and conversation management code. In this case code written for FIPA-OS would need to be restructured.

External DF search was 41% faster than local messaging when using the 115.2Kbps connection, however with 9600 bps it was 242% slower. External FIPA-request was slower than local messaging and with 9600 bps the protocol was 1135% slower. These results indicate that while the external FIPA-request is slower than local messaging the overall system may benefit from accessing complex functionality remotely, such as DF search. We envisage that an intelligent component manager may, knowing the wireless characteristics and availability of components, decide how and where these components are used.

The local minimal mode messaging is 7.5 times faster than using the 115200 bps external communication and 4 times as fast as messaging using the FIPA-OS API locally. In order to get the best performance, local agents should use the minimal mode and the FIPA-OS API should be used only for compatibility when re-using agent code. Agents that perform heavy computation should be executed on a host with more processing power, especially in the case of a low-end device. Furthermore, about the relationship between the component distribution and the wireless link we can say that the poorer the wireless link, the more the communication on the link should be minimized; meaning that the MicroFIPA-OS should be able to work on its own as much as possible.

The external communication presented in the previous section can be optimized on several layers in the FIPA communication stack. At the interaction protocol layer, the agree-performative could be sent implicitly with the inform-message thus removing one message from the interaction protocol. Looking at the message transport layer, we used an internal sockets-based protocol, which does not have the overhead of HTTP; however, a channeled approach removes the TCP three-way handshake that is required for subsequent messages. This approach is used in NAS. One clearly evident optimization, also implemented in MicroFIPA-OS, is to copy messages for receivers at the last possible moment. A message for multiple receivers should be copied to the receivers at the fixed network side, and similarly a message destined for multiple agents on the terminal should be transmitted as a single message.

4 Conclusions

Current technological progress provides the possibility of using desktop-based software and middleware in improving portability and software deployment time on PDAs and other devices. In this paper we examined the performance of agent messaging both within an agent execution environment and between two agent execution environments on PDA devices. MicroFIPA-OS supports the execution of FIPA-OS agents with minor modifications on small devices. The price of portability is an increased demand for system resources and reduced performance. Our results indicate that Java based software needs to be relatively simple in order to gain reasonable execution performance. Executing parts of the functionality on some other node of the network can relieve the performance bottlenecks of the devices. We plan to continue to investigate agent middleware performance and adaptability based on the agent execution context on PDAs and other small devices.

REFERENCES

1. Bergenti F., Poggi A. LEAP: A FIPA Platform for Handheld and Mobile Devices. In Proc. Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL 2001), Seatle, CA, 2001
2. Compaq Research Laboratories. iPAQ H3600 Hardware Design Specification - Version 0.2f. Available at http://www.handhelds.org/Compaq/iPAQH3600/iPAQ_H3600.html
3. Foundation for Intelligent Physical Agents (FIPA). The FIPA Architecture, Geneva, Switzerland, 2001. Available at http://www.fipa.org
4. Foundation for Intelligent Physical Agents (FIPA). FIPA Request Interaction Protocol Specification. Available at http://www.fipa.org/specs/fipa00026/
5. Helin H., Laukkanen M. Towards efficient and reliable agent communication in wireless environments. In Matthias Klush and Franco Zambonelli, editors, Cooperative Information Agents V, Proceedings of the 5th International Workshop CIA 2001, number 2182 in Lecture Notes in Artificial Intelligence, pages 258-263. Springer-Verlag: Heidelberg, Germany, September 2001
6. IONA. Orbix/E 2.1 homepage, 2002. Available at: http://www.iona.com/products/orbix-e.htm
7. Jennings, N., Sycara, K., Wooldridge, M. A Roadmap of Agent Research and Development, Autonomous Agents and Multi-Agent Systems (275-306), Kluwer Academic Publishers, Boston, 1998
8. Laukkanen, M., Tarkoma, S., Leinonen, J. FIPA-OS Agent Platform for Small-footprint Devices. In John-Jules Meyer and Milind Tambe, editors, Pre-proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001), pages 314-325, August 2001
9. Laukkanen M., Helin H., and Laamanen H. Supporting Nomadic Agent-based Applications in FIPA Agent Architecture. In Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2002), July 2002
10. Object Management Group (OMG). The CORBA 2.4 Specification, 2000. Available at ftp://ftp.omg.org/pub/docs/formal/00-11-03.pdf
11. Pendragon Software Corporation. CaffeineMark 3.0 homepage, 2001. Available at http://www.pendragon-software.com/pendragon/cm3
12. Poslad S., Laamanen H., Malaka R., Nick A., Buckle P. and Zipf, A. CRUMPET: Creation of User-friendly Mobile Services Personalised for Tourism. Proceedings of: 3G 2001 - Second International Conference on 3G Mobile Communication Technologies. 26-29 March 2001. London, UK. http://conferences.iee.org.uk/3G2001/
13. Poslad, S., Buckle, P., and Hadinham, R. The FIPA-OS agent platform: Open Source for Open Standards. Proceedings of PAAM 2000, Manchester, UK, pp355-368, 2000
14. Rao H., Chen Y., Chang D., Chen M. iMobile: A Proxy-based Platform for Mobile Services. The First ACM Workshop on Wireless Mobile Internet (WMI 2001), Rome, July 2001
15. Softwired. iBus//Mobile 2.1 Homepage. Available at http://www.softwired-inc.com/products/mobile/mobile.html
16. SourceForge. MicroFIPA-OS and FIPA-OS page at SourceForge open-source repository. Available at http://sourceforge.net/projects/fipa-os/
17. Sun Microsystems. PersonalJava technology White Paper, version 1.2, 2000. Available at http://java.sun.com/products/personaljava/
18. Sun Microsystems. Java 2 Micro Edition homepage, 2001. Available at http://java.sun.com/j2me/