AgentService in a hand[1]

A. Passadore, A. Grosso, M. Coccoli, A. Boccalatte, *Università degli Studi di Genova, Dipartimento di Informatica, Sistemistica e Telematica, Via all'Opera Pia 13, 16145 Genova.*

Abstract—In this paper we present AgentService Mobile: an infrastructure aimed to the execution of agents on devices with limited resources. The mobile device plays the role of a client which consumes a set of services exposed by the AgentService platform. This is the entry of AgentService in a SOA context, with the main goal to open the multi-agent platform to the outside, by using the most recent service oriented technologies.

Keywords: mobile devices, SOA, Web Services, multi-agent systems.

I. Introduction

AGENTS are often in the common imaginary a sort of digital *alter-ego* of the human owners. They play the role of secretaries, avatars, and every other responsibility that can be delegated to a software entity running on a computer. If in the 70's and 80's a computer was a heavy box that rarely went out of offices and labs, since 90's the advent of notebooks allowed users to use their software more or less everywhere.

A new frontier of mobility has been reached with wireless networks that cut off the cables from portable computers. At now the mobility has reached incredible levels with concentrates of computer technology in the palm of a hand as PDAs (Personal Digital Assistants) and cellular phones are.

By using mobile devices we can easily access web pages and interact with remote services, from basilar mail servers to a plethora of web services that remotely expose functions and data. If we hold our all-day life and work activity in a hand, the possibility to host software agents, so close to us, is a real opportunity of releasing them from the closed environment of a multi-agent platform running on a motionless computer.

It is then our goal to expand the range of the agents that are in execution on the platform we are developing: *AgentService* [1].

For this reason, in this paper we present our on-going project with the aim of executing AgentService agents on devices having limited resources, in terms of CPU, RAM and display capabilities.

From a general point of view, this project introduces AgentService in a *Service Oriented Architecture* (SOA) context [2], where the platform exposes its services to the outside, in order to allow remote applications and users to exploit them. Therefore, by using a SOA interface, we do not open AgentService only to C# based agents on mobile devices, but also to whatever software entity able to manage web services. As we will see in section IV, the only other existing solution is based on an internal protocol which enables just compatible agents to contact the remote platform. Our contribution is then aimed to equip the .NET development community with a framework for integrating handheld devices in an agent based system, without renouncing to the flexibility and the opening to the outside.

Following the essence of AgentService, the infrastructure of the system in question is based on the latest technologies in the Microsoft .NET field (the framework is compatible with the Microsoft official releases, and the open source versions, as Mono), in particular the contracted version of the Framework .NET 3.5 (namely the *Compact Framework*) [3] and the *Windows Communication Foundation* (WCF) [4]: a communication infrastructure for building and running connected systems that offers a SOA implementation for Windows-based programming.

We consider AgentService as an alternative to the java based multi-agent platforms, taking into account the absence of solutions in the .NET community. AgentService is especially aimed to industrial applications, where the .NET Framework has gained a conspicuous space.

Considering the limited environment of a mobile device, the main issue of this project was the creation of a light infrastructure for executing agents and exploiting the services offered by a remote standard AgentService platform. AgentService is normally based on the Framework .NET 3.5 and exploits functionalities which have been removed in the Compact Framework version. For this reason, the challenge was the reduction and adaptation of the AgentService libraries to a minimal infrastructure, running on the mobile devices only the essential services and masking the others by using a proxy that contacts the main remote platform through web services. The final goal is to execute on a mobile device an agent that is developed for a standard AgentService application, furnishing a context that apparently is the same of a platform running on a heavy motionless computer.

In this paper we first introduce the main problems related to the development of SOA mobile applications, illustrating the state-of-the-art in relation with the Compact Framework, the Windows Communication Foundation and the other existing solution in the field of multi-agent systems.

[1] A special acknowledgement to Matteo Sommariva, for his precise work during his Master Thesis activity.

In the second part of this paper we show the architecture of the proposed system, focusing on the two main aspects: the runtime service proxies and the light execution environment.

Conclusions, impressions, and a brief comparison with the existing solution will follow.

II. MOBILE (AGENT) APPLICATIONS

A. The vivacious field of mobile applications

In the last years the software development for mobile devices has been subject to a sudden growth due to the massive diffusion of handheld computers. There exist different types of portable devices whose classification is now difficult because of the overlap of their features. However we can outline a brief classification of such devices, distinguishing:

- *Smartphones*: cellular phones with memory and computational resources, mainly aimed to communication features.
- *PDA* (Personal Digital Assistant): devices for the management of personal information as agenda, calendar, block-notes, etc.
- *Tablet PC*: denoting a touch-screen display, it is mainly used in industrial field, but also for gaming and multi-media player.

Smartphones and PDAs are often very similar and easily indiscernible; this fact proves that the trend is to converge to a multi-purpose mobile device with a heterogeneous set of features.

The development of multi-agent mobile applications cannot leave aside the hosting operating system. In the field of handheld devices, there are heterogeneous solutions that make this panorama more variegated than the desktop computers one.

The market is dominated by *Symbian OS* (65% of the market, in the fourth period of 2007), which outclasses *Windows Mobile* (12%), and *Blackberry OS* (11%). Other noteworthy solutions are based on *iPhone OS*, *Palm OS*, *JavaFX*, *OpenMoko Linux*, and the incoming *Google Android*. Some of them are open to third-part applications and provide a SDK for the development in proprietary languages, Java, C++, or C#. Several operating systems support Java libraries (as Blackberry, Android, JavaFx, OpenMoko, and Windows Mobile) and MIDP (Mobile Information Device Profile): a specification for the use of Java on mobile devices; it is part of J2ME: the micro edition of the Java Framework. The Microsoft's framework for handheld devices is the Compact Framework .NET 3.5 which at the present moment is supported only by the Windows Mobile family[2].

The choice of the Windows Mobile OS and the Compact Framework for our project is bind to the fact that AgentService is completely based on the Common Language Infrastructure (CLI) specification [5]. Although the most supported platform is Java, the .NET counter-party denotes features that are in step with the former one[3].

B. SOA Middleware

Mobile devices well represent the role of clients which exploit and consume services hosted on machines with more computational resources. In this sense, the power of a collaborative network involving mobile clients is just the possibility to access to a theoretically infinite set of services from a device with a small CPU and few RAM.

The SOA philosophy embodies the approach we want to apply to AgentService Mobile.

SOA [2] [6] is essentially a sort of style, paradigm, or concept aimed to support services on the web, in order to satisfy the user's requests and consider the single applications as coordinated components of a business process. SOA is based on few strong principles that are strongly relevant in agent based system too:

1) *loose coupling:* with a low intensity of connections among the different components, it is possible to easily update or modify a service without upset the rest of the system. This feature warrants the system scalability and decentralization.
2) *Heterogeneousness:* the different components of a system can be based on different platform and different implementation.
3) High interoperability: an easy communication among the components of the system is a basis of each implementation of both SOA and agent based systems.

The constitutional elements of a SOA are the *services*. A service can be described by three aspects: the *interface* that furnishes the signature of the exposed methods; the *contract*: a formal representation of the interface (expressed, for example, in WSDL); the *implementation* of the service, in strict accordance with the contract. In this sense, an AgentService platform will expose contracts for the usual FIPA services, allowing our mobile agents or other external applications (eventually based on different platforms) to consume them.

SOA is an abstract concept that must be implemented. At now, there exist different solutions which can be classified, considering the way followed by users in order to invoke services [7]: *remote procedure calls* (RPC) or *messages*. Regarding RPC, the client obtains the service interface and then contacts the service method by a (generally synchronous) parametric call. Example of RPC technologies that could be used to implement SOA are CORBA [8], Java RMI [9], and .NET Remoting [10].

[2] Even if there exists an open source version of the Framework named Mono, there is not yet an open source version of the Compact Framework. The developers of Mono ensures the compatibility only for those applications which have no GUI.

[3] There are several comparisons between J2ME and Compact Framework, but almost all are evidently prejudiced. For a equilibrated comparison see: http://www.must.edu.my/~dwong/resources/mobile_commerce_web/j2mevsnetcf.html

The second type is based on the exchange of few well-defined (generally asynchronous) messages. The frameworks based on messages are named *Message-Oriented Middleware* (MOM) and provides a complete management for message queuing, persistence, and security. MOMs support different platforms, protocols, standards, and languages. Because of their versatility and interoperability, they are closer to the SOA concepts. Several big enterprises invest in MOMs and deliver powerful frameworks: *IBM Websphere MQ*, *Sun Java Message Service*, *Microsoft Message Queue Server*, *BEA System Inc* (now Oracle) *MessageQ*, etc.

A technology which matches the two styles and represents a *de facto* standard for SOA system is the one offered by Web Service standards (WS-*) [2] [11]. A web service is based on SOAP: an XML protocol which supports both RPC and messages. Also a Web Service has a contract which is described by WSDL which, like SOAP, is an XML-based language. The combined use of SOAP and WSDL allows the definition of complex, dynamic, versatile systems, which can be easily considered platform-agnostic. Following this paradigm, AgentService Mobile is designed in order to exploit the powerfulness of Web Services. The implementation of the presented project is based on the new framework, introduced by Microsoft, for the development of connected services, named Windows Communication Foundation.

C. Exploiting Windows Communication Foundation in AgentService Mobile

With the release of the Framework 3.0, Windows applications can be based on four sub-systems that manage the different aspects of a software project: *Windows Presentation Foundation* (the graphical layout), *CardSpace* (management of digital identities), *Windows Workflow Foundation* (a complete API for workflow management), and *Windows Communication Foundation*.

The aim of WCF is to provide an environment for the development of distributed applications in Windows-based systems. It is a tool for implementing and hosting services and it supports several industrial standards that define interactions among services, type conversions, marshalling, and protocol management.

A classification of the WCF features [3], which demonstrates its adherence to SOA principles, is reported in the following points:

1) *Independent versioning*: adhering to the WS-* standards, WCF services can be developed with different timetables in respect with consumers.
2) *Asynchronous one-way messaging*: it allows an optimal use of the disposable computational resources.
3) *Platform consolidation*: all the different Microsoft communication technologies (RPC, ASMX, Remoting, COM+, and MSMQ) are now unified under WCF that collects all the features of the single solutions.
4) *Security*: WCF supports several security models.
5) *Reliability*: it warrants the safe delivery of messages, dividing the transport protocol from the delivery mechanism. This approach ensures a safe communication also on untrustworthy channels.
6) *Transaction support for atomic operations*.
7) *Interoperability*: WCF can interact with every single past Microsoft communication technology and other solutions that implement WS-* or REST, as Java.
8) *Performance*: different levels of interoperability and performance are supported.
9) *Extensibility*: every aspect of the platform can be customized, according to the application specification: channels, bindings, codings, transports, etc.
10) *Configurability*, with the support of XML configuration files.

Security and *reliability* are interesting features for the AgentService Mobile architecture: secure transactions among agents, over a reliable channel, are essential needs for trustworthy multi-agent applications.

Interoperability does not close the AgentService platform to contribute of external applications, eventually based on different platforms so, a Java version of mobile device-based agents could be developed in these languages and then, freely interact with the AgentService .NET platform (the Sun's Java project named *Tango* ensures a comfortable interoperability with WCF).

Even if WCF represents a good communication infrastructure, it denotes aspects that can be ameliorated. In the next sections we will show the architecture of the developed system, pointing out every difficulty that the WCF has risen. Especially in conjunction with the Compact Framework, from the client point of view, some issues have complicated the development of the mobile device architecture.

In general the combination of WCF and Compact Framework is not yet sufficiently tested and in the developers' community the experience about them is not so deep. For these reasons, the practice ripened during this project has been significant and appreciable.

D. The Compact Framework 3.5

A brief introduction of the Compact Framework is necessary to understand the technical choices that have guided the developers to the actual architecture of AgentService Mobile.

With a dramatic reduction to 30% of available classes in respect to the Framework .NET 3.5 and a physical size of 4 MB, the Compact Framework represents a typical bottleneck for development of complex applications, as the AgentService multi-agent platform is.

With the Compact framework, it is possible to develop applications in C# or Visual Basic .NET. A special high-performance, just-in-time compiler is provided with the framework.

Figure 1 shows which features are preserved in the Compact Framework. Server functionalities, ASP.NET, C++ and J# Development, and Remoting are not supported. In particular, Remoting would have been useful to create a communication channel between the mobile client and the

remote platform, simply calling the remote methods of the modules and services of AgentService.

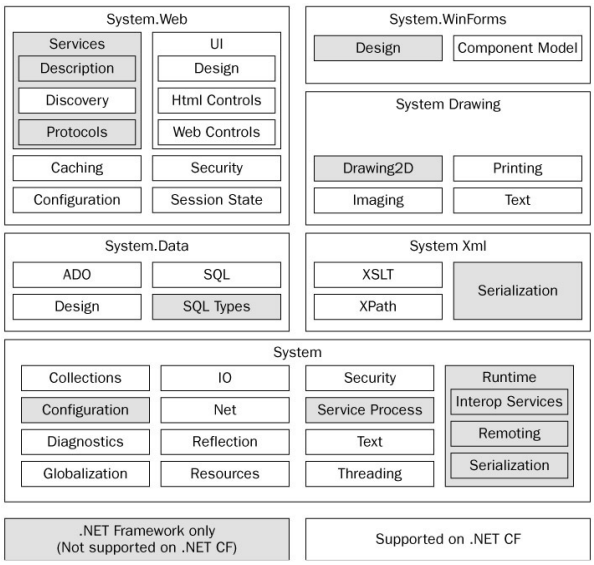

Figure 1: differences between .NET CF and .NET Framework.

Significant restrictions concern the serialization functions (both XML and binary), reflection, and threads. These restrictions have a certain impact during the porting of basic AgentService services from the usual Framework to the Compact one.

Appreciable is the support nearly complete to the Windows Communication Foundation that, in absence of Remoting (and thanks to other considerations that will be clear in the next subsection), is the chosen way to implement a communication channel between mobile agents and the resident platform or agents.

E. Agents on smart devices

In the multi-agent systems research and industry communities, different applications that consider agents on hand-held devices have been suggested. The application domains are quite different: the tourism industry [12], home care service [13], museum guides [14], educational [15], tracking of goods with RFID [16], ambient intelligence [17], industrial maintenance [18], etc.

Most of them exploit the well-known multi-agent framework called JADE [19] along with the LEAP extension [20], especially aimed to the execution of agents on mobile devices. The remaining solutions implement *ad-hoc* architectures essentially based on the Java framework and J2ME. These facts induce two considerations: JADE holds supremacy also in the management of agents on PDAs; if AgentService is one of the very little examples of MAS developed by using the .NET Framework, it is the only one that supports agents on the Compact Framework and then represents a different point of view for mobile agent programming.

Nonetheless, JADE represents once more the state of the art, considering the largeness of the project and the employed resources. For this reason we consider JADE LEAP as a basis for comparison and we report in the next paragraph a short introduction to the architecture.

F. JADE LEAP

LEAP means Lightweight and Extensible Agent Platform. The aim of this international project (involving some big enterprises as Motorola, Siemens, Broadcom, British Telecom, and TILAB) is to reduce the JADE framework in order to execute an agent container in a device with few resources and allow this agent to communicate with other agents running on different containers or platforms. LEAP is executable on every operating system supporting Java, from a powerful server with J2EE (Enterprise Edition), to a smart phone supporting MIDP.

A JADE platform is composed of containers: processes that are in charge to host and execute agents, providing runtime services. These containers must connect to a main container that represents the bootstrap point of the platform and hosts basic services as the *Directory Facilitator* and the *Agent Management System* (AMS). Agents running on different containers or even different platforms can communicate by using the MTP (Message Transport Protocol) furnished by JADE.

LEAP is totally inspired to the JADE architecture, therefore the idea is to execute a LEAP container over the handheld device, maintaining the same API of JADE. The container is connected by a main container residing in a desktop computer. Usually two JADE containers (which could be deployed over a network) communicate by using RMI; due to restrictions of J2ME this is not possible with LEAP, then a new proprietary protocol has been developed: the JICP (JADE inter-container protocol) which makes incompatible JADE and LEAP containers.

An appreciable feature of LEAP is the possibility to split the container into a front-end running on the device and a back-end running on a remote computer. The goal is to lighten the device, hosting almost all the runtime services on the back-end.

III. AGENTSERVICE MOBILE EDITION

A. Overview

Leaving aside a complete introduction of AgentService (for an exhaustive overview read [21]) we concentrate only on those elements that are essentials for the execution of mobile agents.

First, AgentService provides a particular model where an agent is essentially composed of *behaviours* and *knowledges*. Behaviours represent the business activities of an agent and generally are managed as threads executed concurrently. Behaviours of the same agent can share information by using knowledge objects: a sort of knowledge base containing data that can be persisted and that must be accessed concurrently. AgentService Mobile keeps the same model because is a specific requirement the possibility of running standard agents on a mobile device.

From the agent point of view, the AgentService platform is a sort of operating system that exposes services through a

runtime interface, dispatches messages, and schedules the agent behaviours.

The *runtime* interface is available from any agent behaviour. It exposes services useful during the execution time of a behaviour:

- *Yellow pages*: as suggested by FIPA, it is directory for publishing and advertising the services managed by the agent.
- *White pages service*: a sort of telephone directory for retrieving the agent addresses.
- *Console*: a service for monitoring the execution of an agent through text messages.
- *Context*: it returns the behaviour execution context, where it is possible to create and start new behaviours and create new knowledge objects.
- *Logging service* for the monitoring of behaviour execution.
- *Message service*: it is responsible for the forwarding of messages to the message module (and then to the recipient queue). Moreover, it supports the instantiation of conversations, namely preferential communication channels between two behaviours of different agents.
- *Mobility*: an infrastructure for the migration of an agent from a platform to another one.
- *Persistence*: in order to preserve the agent status following up a system crash or failure, this service freezes the agent and saves it in a storage facility (databases, xml file, etc...).

Considering the AgentService Mobile project, the majority of these services should reside on the remote platform. Agents that run on a mobile device access them through a proxy that hides the communication channel with the remote server. Other services, as persistence and logging, must be local, in order to maintain the information regarding agents on the local mobile machine. Finally, the mobility service is unsupported due to the purpose of these types of agent that are intimately tied with the mobile device.

Another fundamental aspect that AgentService Mobile must take into account is the *message dispatching*. Usually, an agent which wants to send a message to a peer, contacts, through the message client provided by the *runtime*, the messaging module which delivers the message to the message queue of the recipient agent. In case of the recipient is running on a federated platform, the messaging module directly contacts the other messaging module through the .NET Remoting. Unluckily, this simple mechanism is not allowed by the Compact Framework, therefore a mobile agent cannot directly interact with the remote central messaging module but must pass through a proxy, as shown in the following.

Incidentally, a recent improvement of the messaging module made it faster by suppressing every polling loop for checking the message queue, in waiting for a new message. Actually, every behaviour-thread that is waiting for a message, is put in *waiting* status and released only when an event, notified by the messaging module, occurred[4]. This enhancement increases the speed up to 350 times. The use of this approach allows also keeping down the resources consuming in the mobile version, where this requirement is fundamental.

Last, the scheduler is in charge of managing the agent behaviours running them as threads. In the standard AgentService version there are different policies to execute behaviours:

- *One behaviour, one thread*: this is the preferred way in term of performance.
- *Every behaviour in a single thread*: preferable in term of resource occupation.
- A *mixed approach* for scenarios where some behaviours have to be executed taking into account the performance and others, the resources.

Due to limitations of the Compact Framework in thread management, we chose to implement the simplest scheduler, namely the first. The tests we made, both on an emulator and on physical devices gave a good response in term of execution speed.

B. Architecture

Figure 2 shows the whole architecture involving the mobile sub-platform and the remote, motionless AgentService system.

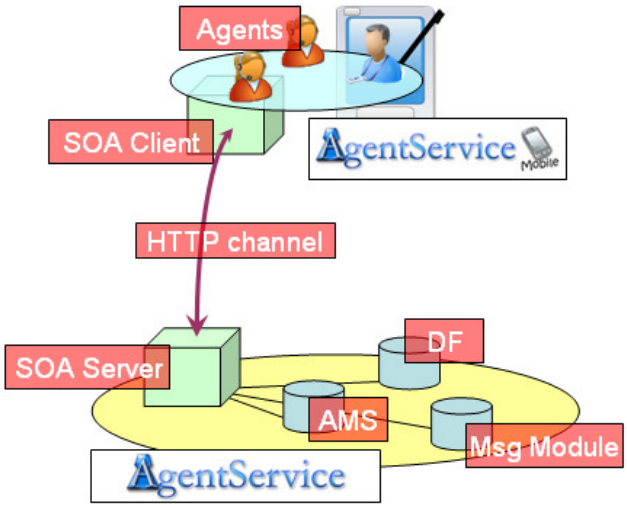

Figure 2: AgentService Mobile architecture.

From the server-side, AgentService deploys a SOA server which receives the requests from mobile agents. It is a sort of bridge between the mobile device and the basic services exposed by AgentService. At the communication level it creates an *http channel* (the only one supported by the Compact Framework) and sends SOAP messages. It

[4] for further information, see *AutoResetEvent* on http://www.developerfusion.co.uk/show/5184/3/

uses a bidirectional message exchange, in order to simulate a remote calling of a method (*invocation* and *return value*).

The AgentService SOA server has been implemented to reserve the channel for a short time, in order to avoid pending calls. For example, the waiting for a message: between the call of *WaitForMessage* method and its return, several minutes could pass; for this reason the SOA server does not wait for the message, leaving the call pending, but it is the mobile client that periodically contacts the server (See **Figure 4**).

Essentially, the SOA Server it is a sort of special runtime that is able to contact the messaging module, the AMS, and the DF, on mobile agents place.

Moreover, it manages the logging credentials for accessing the server from a mobile device. From a configuration file, it can be possible to set up a platform opened to everyone, or a protected one, based on a list of credentials.

Finally, the server periodically controls the connection state of the mobile devices. In case of timeout (customizable in the configuration file), it deregisters the lost mobile agents from the AMS and DF.

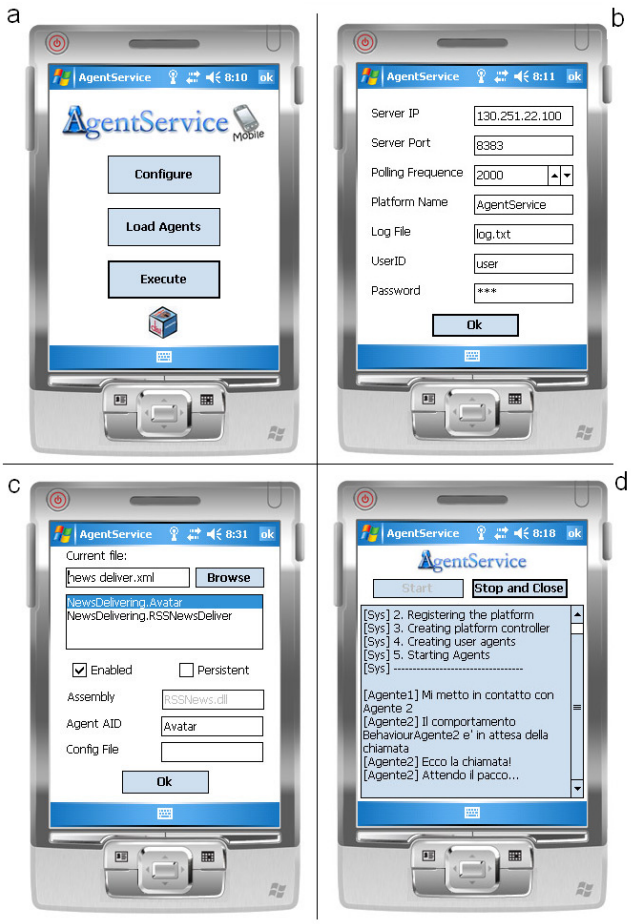

Figure 3: a) the main form of AgentService Mobile; b) general settings; c) batch for agent execution; d) the console.

On the client side, the architecture is implemented by an executable file which instantiates and executes the agents the user suggests in a configuration file or through the GUI.

Screenshots in **Figure 3** show the settings (**Figure 3**.b and **Figure 3**.c) the user must enter in order to connect to a remote platform (settings that can be stored in a configuration file). In particular, he must set the IP address and the port through which the SOA Server is listening and he must enter the username and password that allow him to gain the access to the platform. An interesting setting is the polling time which indicates how frequently the polling thread monitors events coming from the remote platform. These events regard, for example, the notification of a new incoming conversation request, the presence of a new message, etc. This polling loop is the only one running on the mobile platform: it is an important technical solution, because we avoid the saturation of the CPU due to a looping thread for each behaviour. Thanks to this solution, behaviours that are waiting for a message, notify their request to the polling loop manager and then put their thread in *waiting* status. As illustrated in **Figure 4**, when a notification of an incoming message arrives to the polling thread, the behaviour is awoken, the message is downloaded, and the behaviour execution can continue. It is important to mark that from the agent point of view, this complex procedure is hidden behind a simple method call.

C. AgentService Mobile at runtime

The mobile side of the presented solution consists in a *driver* object that first runs the SOA client, contacting the SOA server and presenting the user credentials. The SOA client also runs the aforementioned polling thread and creates a *runtime* object: a sort of proxy for the remote services hosted in the standard AgentService platform. Second, the *driver* executes the *mobile platform controller* which is in charge of creating agent instances, taking the information about the types of agent templates, behaviours and knowledge objects by using the .NET Reflection. The platform controller also binds the runtime object to the agent instances created by the SOA Client, granting a solid link to the services of the remote platform, in a totally transparent manner for the mobile agents. Finally, the platform controller starts the agents, registering them to the remote AMS and activating the behaviour schedulers.

Figure 5 describes in details the bootstrap process, from the start of the driver to the agent execution. It shows the sequences of steps that allow mobile agents to subscribe to the remote platform services. From the UML diagram transpires the most important feature of the *driver* which eases the agent activities furnishing a runtime environment identical to the desktop platform one.

IV. CONCLUSIONS

A. Implementing SOA with WCF

In the previous sections we shown the technical solutions adopted to implement AgentService Mobile, applying the SOA model to a multi-agent system infrastructure. WCF offers a particular point of view, largely diffused in the Windows-based programming.

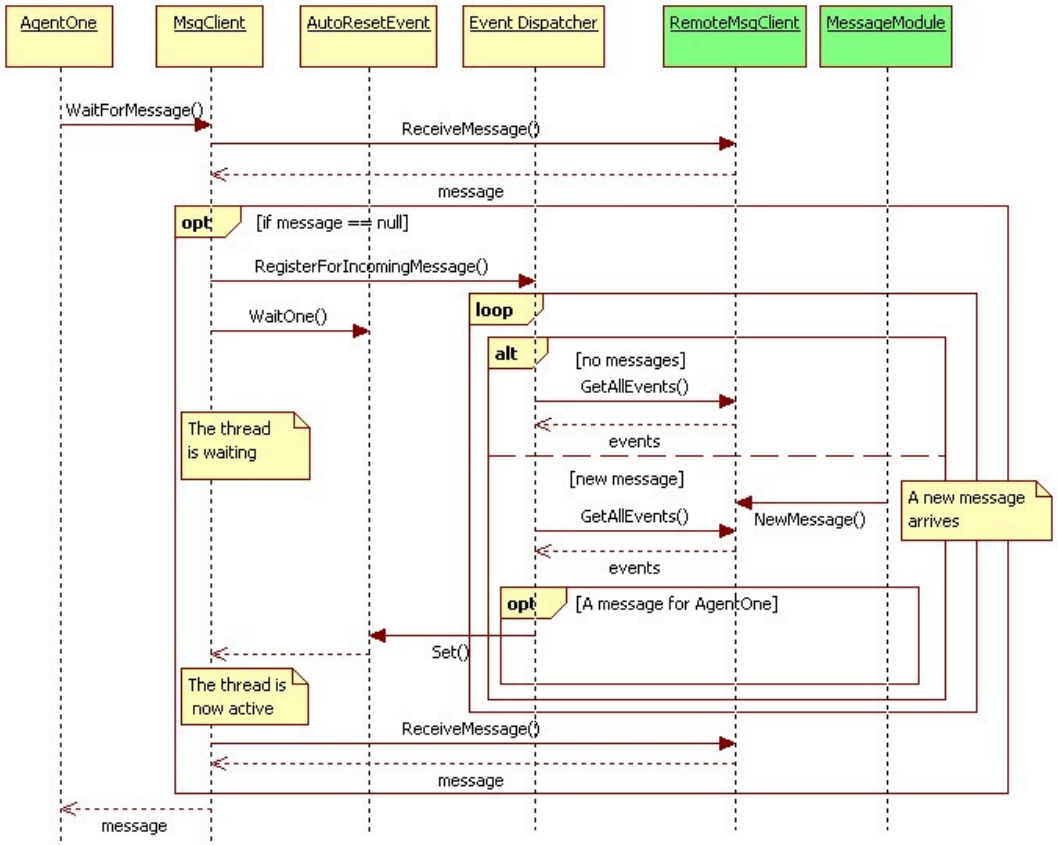

Figure 4: how an agent gets a message.

The good points appeared clearly in the previous sections, but during the implementation emerged some points which can be considered as faults. In particular, the integration of WCF and Compact Framework, although appreciable and powerful, represent a slight difficulty in order to exploit the real comfort of web services and WCF programming. In particular, due to the sensible reduction operated in the Compact Framework, the WCF support is surely complete but free from those classes that represent a comfortable surround. For this reason, because the WCF on the Compact Framework does not directly support the *Data Contract serialiazer* (the standard WCF serializer), we had to manually instruct the *basic xml serializer* in order to (de)serialize the messages used to invoke the WCF services.

A further issue regards not only WCF but also the SOA paradigm. It is a usual need, among agents, to send objects as message content. For this reason the message body is defined in AgentService as a box which can contain instances of every type targeting the .NET Framework (and then which derives from the *System.Object* class). The rigidity of the contract which characterizes every web service does not permit the invocation of a method passing a parameter which contains a generic object. This lack of flexibility forces us to convert such generic objects in strings containing their binary serialization, granting a solid contract which counts strings instead of generic objects.

B. JADE LEAP and AgentService Mobile

For the two projects, the main obstacle was the porting of the platform infrastructure on a little device. Curiously, for both the implementations, the principal cause was the message transport sub system. In JADE, agent containers exchange messages by using the Java RMI RPC technology. In AgentService does not exist a correspondence with containers, because we consider a platform the basic environment for circumscribed agent communities. In order to interconnect different agent societies we use .NET Remoting (which can be compared with Java RMI). Both Java RMI and .NET Remoting are not supported by J2ME and Compact Framework, here-hence the need to implement a new way for message exchanging. In JADE LEAP they opted for the creation of a proprietary protocol named JICP. In AgentService we opted for a service-oriented solution.

Continuing to analyze the differences between JADE LEAP and AgentService, because of the existence of the container concept in JADE, LEAP developers created a complete and autonomous infrastructure similar to a usual container to host agents on a mobile device. In AgentService we implement a minimal infrastructure which essentially warrants the execution of an agent and entrusts the implementation of each service to a remote standard

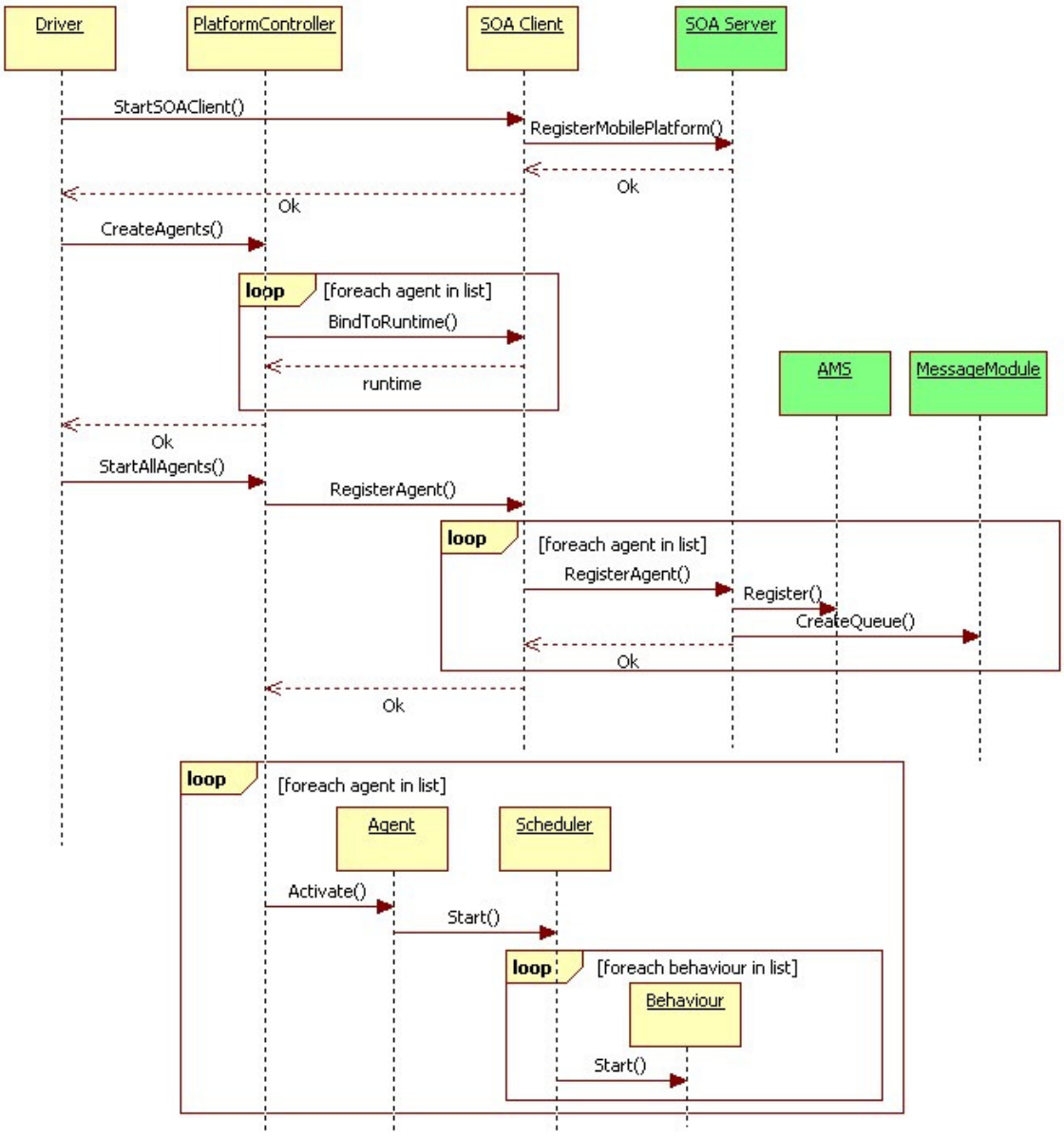

Figure 5: the bootstrap process.

AgentService platform. The JADE LEAP solution guarantees a certain independence for an agent container. On the other hand, the AgentService Mobile solution reduces the resources consuming of the infrastructure, leaving computational resources to the agent application. Our approach looks like the split container option in JADE LEAP but, while in LEAP it seems to be discouraged in the majority of cases, in AgentService Mobile gives good results in term of speed and performances.

A further difference between the two solutions is the fact that LEAP is an add-on, while AgentService Mobile is embedded in AgentService. In case the user want to add mobile agents on a preexistent platform, in AgentService it is sufficient to enable the SOA Server, while in JADE the platform must be replaced with the LEAP version, because a LEAP container and a JADE container must not live together (the cause is the different message transport sub system). From this point of view the AgentService solution appears to be more flexible.

Moreover, the service oriented interface of AgentService is not only aimed to agents on mobile devices but also to those external applications which want to communicate with agents and exploit the platform services.

We experienced that this requirement is fundamental for a lot of AgentService users.

C. Future works

AgentService Mobile is an ongoing project which can be improved in order to provide a more useful environment for our agents. Considering the compliancy to web standards, AgentService follows the *ws-** specification, but additional work as to be done for supporting *ws-security*, *ws-reliability*, and *ws-atomic transaction* that could be employed in the agents interaction protocol infrastructure

provided by AgentService in both the design and execution phases.

In addition, we will develop a light message dispatcher embedded in the mobile infrastructure, in order to avoid the employ of the remote message module for those conversations which involve agents running on the same device. This modification could distort the essence of our project, because every platform service is now considered as a web service. Nonetheless, for some application the rate of messages exchanged among agents hosted in the same device could justify this by pass. For this reason, the local message dispatcher will be optional.

An interesting evolution that confirms the trend to run agents on heterogeneous and limited devices is the development of an ultra-light infrastructure to host an agent on an embedded device. Recently, a .NET Micro Framework [21] has been delivered, in order to export the .NET programming also on basic devices with ARM processors, few RAM and no operating system. It could be interesting to run micro agents on these small devices and connect them to a standard AgentService platform. This improvement could consolidate AgentService applications in the industrial field, with agents deployed on sensors and actuators.

REFERENCES

[1] C. Vecchiola, A. Grosso, A. Boccalatte; "AgentService: a framework to develop distributed multi-agent systems", *Int. J. Agent-Oriented Software Engineering*, Vol. 2, No.3 pp. 290 – 323, 2008.

[2] N. Josuttis, "SOA in Practice, The Art of Distribuited System Design", O'Reilly, 2007.

[3] J. Smith, "Inside Microsoft Windows Communication Foundation",Microsoft Press 2007

[4] J. Löwy, "Programming WCF Services", O'Reilly Media, 2007

[5] Standard ISO/IEC 23271:2003: Common Language Infrastructure, March 28, 2003, ISO.

[6] OASIS, "Reference Architecture for Service Oriented Architecture 1.0", Public Review Draft 1, Apr. 23, 2008, http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-pr-01.pdf

[7] D. Melgar, "Message-Centric Web Services vs RPC-Style Invocations", *SOA World Magazine*, March 2003.

[8] Object Management Group, CORBA 3.0 Specification http://www.omg.org/technology/documents/formal/corba_2.htm

[9] Sun Developer Network, Remote Method Invocation (RMI) http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp

[10] NET Framework Developer's Guide, .NET Remoting Overview http://msdn.microsoft.com/en-us/library/kwdt6w2k.aspx

[11] W3C Working Group, Web Services Glossary, 11 February 2004 http://www.w3.org/TR/ws-gloss/

[12] J. S. Lopez and F. A. Bustos, "An Agent Application on the Tourism Industry", In *Proceedings of the International Joint Conference IBERAMIA/SBIA/SBRN 2006 - 1st Workshop on Industrial Applications of Distributed Intelligent Systems (INADIS'2006)*, Ribeir˜ao Preto, Brazil, October 23–28, 2006

[13] G. Itabashi, M. Chiba, K. Takahashi, Y. Kato, "A Support System for Home Care Service Based on Multi-agent System",In *Proceedings of Fifth International Conference on Information, Communications and Signal Processing*, 2005

[14] M. Bombara and D. Cal and C. Santoro, "Kore: A multi-agent system to assist museum visitors", In *Proceedings of the Workshop on Objects and Agents (WOA2003)*, 2003

[15] E. McGovern , B. J. Roche1 , E. Mangina and R. Collier, "IUMELA: A Lightweight Multi-Agent Systems Based Mobile Learning Assistant Using the ABITS Messaging Service", Lecture *Notes in Computer Science*, Springer Berlin / Heidelberg, 2007

[16] Feng Li Ying Wei, "Tracking In-Transit RFID-Tagged Goods Using Multi-Agent Technology", In *Proceedings of International Conference on Wireless Communications, Networking and Mobile Computing, WiCom 2007*.

[17] D. I. Tapia, J. Bajo, J. M. Sánchez and J. M. Corchado, "An Ambient Intelligence Based Multi-Agent Architecture", *Developing Ambient Intelligence*, Springer Paris, 2008

[18] A. Passadore and G. Pezzuto, "A multi-agent platform supporting maintenance companies on the field", In *Proceedings of the Workshop on Objects and Agents (WOA2007)*, 2007

[19] F. Bellifemine, G. Caire, D. Greenwood, "Developing Multi-agent Systems with JADE", John Wiley & Sons, 2007.

[20] M. Berger, S. Rusitschka, D. Toropov, M. Watzke, M. Schlichte, "Porting Distributed Agent-Middleware to Small Mobile Devices", In *Proceedings of the Workshop on Ubiquitous Agents on embedded, wearable, and mobile devices*, Bologna, 2002.

[21] D. Thompson, R. S. Miles, "Embedded Programming with the Microsoft .NET Micro Framework", Microsoft Press, 2007.