Whitestein Series in Software Agent Technologies

This series reports new developments in agent-based software technologies and agent-oriented software engineering methodologies, with particular emphasis on applications in various scientific and industrial areas. It includes research level monographs, polished notes arising from research and industrial projects, outstanding PhD theses, and proceedings of focused meetings and conferences. The series aims at promoting advanced research as well as at facilitating know-how transfer to industrial use.

About Whitestein Technologies

Whitestein Technologies AG was founded in 1999 with the mission to become a leading provider of advanced software agent technologies, products, solutions, and services for various applications and industries. Whitestein Technologies strongly believes that software agent technologies, in combination with other leading-edge technologies like web services and mobile wireless computing, will enable attractive opportunities for the design and the implementation of a new generation of distributed information systems and network infrastructures.

www.whitestein.com

Software Agent-Based Applications, Platforms and Development Kits

Rainer Unland
Matthias Klusch
Monique Calisti
Editors

Birkhäuser Verlag
Basel · Boston · Berlin

Editors:

Rainer Unland
Universität Gesamthochschule Essen
Fachbereich Mathematik und Informatik
Schützenbahn 70
D-45117 Essen

Matthias Klusch
German Research Center for Artificial Intelligence,
Deduction and Multiagent Systems
Stuhlsatzenhausweg 3
D-66123 Saarbrücken

Monique Calisti
Whitestein Technologies
Pestalozzistrasse 24
CH-8032 Zürich

2000 Mathematics Subject Classification 68T35, 68U35

A CIP catalogue record for this book is available from the Library of Congress, Washington D.C., USA

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

ISBN 3-7643-7347-4 Birkhäuser Verlag, Basel – Boston – Berlin

ISBN-10: 3-7643-7347-4
ISBN-13: 978-3-7643-7347-4
e-ISBN: 3-7643-7348-2

9 8 7 6 5 4 3 2 1
www.birkhauser.ch

Contents

Preface

Intelligent agents and multi-agent systems (MAS) represent the next big step in the development of next-generation software systems, especially when considering large scale distributed applications consisting of several sub-components with behavior that is increasingly difficult to predict. This is supported by important research and development results and reinforced by the increasing uptake of agent-based solutions and services for real-world industries. In fact, software agent technology successfully addresses a number of highly relevant issues, like efficient resource distribution, scalability, adaptability, maintainability, modularity, autonomy, self-sustainability, and decentralized control, by providing powerful concepts, metaphors and tools. The mentioned issues are often regarded as essential non-functional properties of emerging software architectures and systems.

The high importance of agent-related research and development can be seen from the fact that currently about 100 major projects are funded in Europe only - see http://www.agentlink.org/resources/agentprojects-db.php - and more than 100 academic and commercial software tools are publicly advertised - see http://www.agentlink.org/resources/agent-software.php. And these numbers are still growing. As a result of the enormous efforts the stage of maturation has reached a level, which encourages commercial players to increasingly adopt multi-agent systems concepts and technologies for the development of a variety of real-world applications in different domains such as logistics, e-commerce, and entertainment. In this perspective, concrete agent-driven research and development results (such as applications, platforms, and development kits) substantially contribute to promote the technology and increase its exploitation for industrial solutions.

This book provides a first and comprehensive overview of existing software agent development kits, environments, and applications. It is intended to be of particular use for those who want to assess the maturity and state-of-the-art of applied software agent technology. Both the software engineering and the user perspective are covered by a carefully selected set of contributions reporting on prominent examples of agent development environments, platforms, and toolkits and deployed agent-based applications from various different application areas. In particular, most of them have been either successfully demonstrated to the public at the agent technology exhibition of the first German conference for Multi-Agent system TEchnologieS (MATES 2004) in Erfurt, or won the prestigious system innovation award of the international workshop series on Cooperative Information Agents (CIA). Since this book concentrates on implemented systems most of them are available on the Internet. Thus, at the end of each paper you will find all relevant information about where to get the software on the Internet (if it is available) and whom you may contact in case of questions.

Contents

The book consists of seven chapters. The assignment of papers to chapters has been a hard choice, since many papers fall into several categories. However, we believe the final layout is the most reasonable one.

The first three chapters (with eight papers altogether) present toolkits for the development of multi-agent systems. The toolkits are subdivided into three categories: platforms, development environments, and frameworks. An *agent platform* is intended as the set of middleware components supporting the development of (distributed) multi-agent applications. It provides all basic services, like agent life-cycle management, communication, tasks scheduling, security, etc., to easily initialize and run multi-agent systems. A *development environment* usually supports all phases in multi-agent system engineering, which comprises requirements engineering, system design, development and deployment. *Agent frameworks* provide a high-level programming environment consisting of a multi-agent system skeleton that allows the programmer to easily extend it to a full-fledged MAS application. Toolkits can also be differentiated according to their focus of support. In general, it is possible to distinguish between middleware- and reasoning-oriented systems. In this latter case, one emphasizes rationality and goal-directedness support for agent development.

The first chapter focuses on agent platforms and starts with the paper The *JADE Platform and Experiences with Mobile MAS Applications* by Fabio Bellifemine, Giovanni Caire, Giosuè Vitaglione, Giovanni Rimassa, and Dominic Greenwood. JADE is a well-known and well-established Java-based and FIPA-compliant agent platform. The paper gives a comprehensive overview about the basic concepts of JADE. Furthermore, it shows how JADE can be used on mobile networks. Finally, it discusses possible application domains for JADE. The second paper *$\mathcal{A}$-globe: Agent Development Platform with Inaccessibility and Mobility Support* by David Šišlák, Martin Rehák, Michal Pechoucek, Milan Rollo, and Dušan Pavlicek presents $\mathcal{A}$-globe, a streamlined lightweight platform for MAS development, which operates on normal PCs as well as on PDAs. After a comprehensive introduction into the basic features of $\mathcal{A}$-globe it is compared to some other agent platforms. The next section concentrates on simulation support since $\mathcal{A}$-globe provides a special infrastructure for environmental simulation. The third paper in this chapter *Supporting Agent Development in Erlang through the eXAT Platform* by Antonella Di Stefano and Corrado Santoro motivates and presents first the agent programming language Erlang. Then the agent programming platform eXAT that is based on Erlang is discussed. eXAT especially emphasizes the implementation of agent intelligence, behavior, and communication.

The second chapter is dedicated to development environments. The paper *Living Systems® Technology Suite* (LS/TS) by Giovanni Rimassa, Monique Calisti, and Martin E. Kernland describes the LS/TS set of components for the development and deployment of products and systems based on software agent technology

and autonomic computing. The paper not only gives a comprehensive overview about the architecture and functionality of this package, but also discusses the challenges that were to be addressed in order to develop the proposed software methodology and infrastructure. The second paper by Vladimir Gorodetsky, Oleg Karsaev, Vladimir Samoylov, Victor Konushy, Evgeny Mankov, and Alexey Malyshev presents the *Multi-Agent System Development Kit* (MASDK), a comprehensive software tool kit for the development, implementation, and deployment of multi-agent systems. The paper mainly concentrates on the development process, which is heavily influenced by the Gaia methodology. It is conducted with the help of a number of integrated editors (e.g., for the model, protocol, ontology, behavior, and state transition development), which are described in detail. The third paper *An Integrated Development Environment for Electronic Institutions* by Josep Lluís Arcos, Marc Esteva, Pablo Noriega, Juan Antonio Rodríguez-Aguilar, and Carles Sierra presents a methodology and an integrated development environment for engineering multi agent systems as electronic institutions. The latter defines a set of artificial constraints that articulate agent interactions, defining what they are permitted and forbidden to do. It defines a normative environment where heterogeneous (human and software) agents can participate by playing different roles and can interact by means of speech acts. The integrated use of these tools is illustrated using as an example the double auction market.

The third chapter starts with the paper *Jadex: A BDI-Agent System Combining Middleware And Reasoning* by Lars Braubach, Alexander Pokahr, and Winfried Lamersdorf. The presented system Jadex relies on an arbitrary given agent platform, e.g. JADE, however, extends it by providing tools to model agent rationality and goal-directedness. Its reasoning engine supports cognitive agents by exploiting the BDI model. It permits to explicitly model such features as beliefs, plans, goals or capabilities. The CAFnE toolkit is presented in the paper *Component Agent Framework For Non-Experts (CAFnE) Toolkit* by Gaya Jayatilleke, Lin Padgham. and Michael Winikoff. The vision of the authors is not only to support developers in the initial application development but also to provide a framework that facilitates domain experts themselves in making modifications to a deployed system, in order for it to better fit needs which are identified as the system is used. The system is introduced and its functionality is explained with the help of an example system.

The forth chapter comprises two papers that show how Web-Services can be integrated into agent technology. The paper T*he WSDL2Agent Tool* by László Zsolt Varga, Ákos Hajnal, and Zsolt Werner presents a bipartite tool for this purpose. The WSDL2Jade part of the tool generates code for a proxy agent that makes the Web service available in a multi-agent environment. The WSDL2Protégé part of the tool translates a WSDL description to a Protégé project in order to support its semantic enrichment. It generates a project file for the Protégé ontology engineering tool in which the ontology of the Web service can be visualized, edited,

or exported to various formats. The second paper *WS2JADE: A Tool for Run-Time Deployment and Control of Web Services as JADE Agent Services* by Xuan Thang Nguyen, Ryszard Kowalczyk, Mohan Baruwal Chhetri, and Alasdair Grant presents the WS2JADE framework. It permits the easy integration of Web services into the JADE agent platform. In particular, the technical aspects of the run-time deployment and control of Web services as agent services are discussed. The Web service - agent integration capabilities of WS2JADE are demonstrated with simple examples of Web service management including service discovery, composition, and deployment with JADE agents.

Chapter five concentrates on *Tool Support for Agent Communication and Negotiation*. The paper by Tibor Bosse, Catholijn M. Jonker, Lourens van der Meij, Valentin Robu, and Jan Treur presents SAMIN, *A System for Analysis of Multi-Issue Negotiation*. The agents in this system conduct one-to-one negotiations, in which the values across multiple issues are negotiated on simultaneously. The paper shows how the system supports both automated and human negotiation. To analyze such negotiation processes, the user can enter every formal property deemed useful into the system and use the system to automatically check this property in given negotiation traces. The paper also shows how to deal with incomplete information and presents some experimental results about human multi-issue negotiation. F*uzzyMAN: An Agent-Based E-Marketplace with a Voice and Mobile User Interface* by Frank Teuteberg and Iouri Loutchko focuses on the conceptual foundations and the architecture of an agent-based job e-Marketplace that supports mobile negotiations. The negotiation model is based on many negotiation issues, a fuzzy utility scoring method, and simultaneous negotiation with many negotiation partners in an environment of limited negotiation time. The paper discusses FuzzyMAN's architecture, agents, negotiation model, and mobile and voice user interfaces. Heikki Helin and Mikko Laukkanen deal in their paper with *Efficient Agent Communication in Wireless Environments*. They propose a layered model of agent communication in the context of the FIPA agent architecture. For each layer of this communication stack an efficient solution for wireless agent communication is presented. Furthermore, the paper thoroughly analyzes the performance of agent communication in slow wireless environments.

Chapter six contains two papers about tool kits for mobile agents. The paper *AMETAS - The Asynchronous MEssage Transfer Agent System* by Michael Zapf presents a development and runtime environment for creating and running mobile, autonomous agents under Java 2. AMETAS defines three kinds of application components: agents, user adapters, and services. Services are able to wrap system-dependent resource accesses and provide functional enhancements while user adapters integrate the human user into the agent environment. Techniques of mediation are used to realize open applications; i.e. applications with an ever-changing set of components. The discussed security system prevents illegal access between users and defines the access control to resources. The other paper *Tracy: An Extensible Plugin-Oriented Software Architecture for Mobile Agent Toolkits*

by Peter Braun, Ingo Müller, Tino Schlegel, Steffen Kern, Volkmar Schau, and Wilhelm Rossak presents a kernel-based tool kit that only provides fundamental concepts and functions common to all toolkits and abstracts from all of their possible services. In particular, although Tracy was developed as a mobile agent toolkit, its kernel abstracts from all issues related to agent mobility, delegating this to an optional service implementation. This makes it possible to replace Tracy's migration service with another implementation and even to have two different migration services in parallel. Service implementations are developed as plug-ins that can be started and stopped during run-time. The paper first discusses the set of fundamental services. Then it is shown how they are realized in Tracy.

Chapter seven comprises three papers that are related to agent-based applications. Each of these papers also covers a research issue, however, discusses its solution on the basis of an application. *The Packet-World: A Test Bed for Investigating Situated Multi-Agent Systems* by Danny Weyns, Alexander Helleboogh, and Tom Holvoet presents as application area the packet world. The research aim of the paper is to discuss how to model a distributed application as a set of cooperating autonomous entities (agents), which are situated in an environment. The Packet-World is used as a test bed to explore and evaluate a broad range of fundamental concepts and mechanisms for situated MASs. The paper elaborates on the structure of the environment, agents' perception, flexible action selection, protocol-based communication, execution control and timing, simultaneous actions and several forms of stigmergy. *Decommitment in a Competitive Multi-Agent Transportation Setting* by Pieter Jan 't Hoen, Valentin Robu, and Han La Poutre discusses the decommittment issue on the basis of a large-scale logistics setting (freight forwarding) with multiple, competing companies. It is shown in the paper that decommitment as the action of foregoing of a contract for another (superior) offer can reach higher utility levels in case of negotiations with uncertainty about future opportunities. The paper *Teamworker: An Agent-Based Support System for Mobile Task Execution* by Habin Lee, Patrik Mihailescu, and John Shepherdson shows how a multi-agent based computer cooperative support system known as TeamWorker can help to overcome the difficulties faced by mobile workers. Each mobile worker is assigned a personal agent that can assist her/him during the working day through appropriate service provision (based on current work context), and through monitoring work progress to anticipate and undertake required actions on the user's behalf. A detailed presentation of the TeamWorker system is given, including the benefits provided for a real life mobile business process.

As this book is a collaborative effort, the editors would like to thank foremost the contributing authors for their outstanding contributions, and the reviewers and publisher for their invaluable help and assistance during the whole project. We also would like to thank Dr. Stefan Göller from Birkhauser Publishing Ltd. for his outstanding support in producing this book.

In summary, we hope that this book will be of substantial benefit for students, software engineers, computer scientists, researchers (both academic and industrial), and IT experts, who are keen to learn about the deployment of software agent technology for engineering complex solutions and systems.

Enjoy the reading!

Monique Calisti, Matthias Klusch, Rainer Unland
Zürich - Saarbrücken - Essen
Spring 2005

The JADE Platform and Experiences with Mobile MAS Applications

Fabio Bellifemine, Giovanni Caire, Giosuè Vitaglione,
Giovanni Rimassa and Dominic Greenwood

Abstract. This paper draws a perspective about a software platform for multi-agent systems, called JADE. JADE is an Open-Source Java middleware very popular in the MAS research and development community, counting a lively and active user base. Here we will describe the JADE architecture, the main features provided by the platform, and some application domains from the Open Source community.

1. Introduction

JADE [1] is a middleware for the development of distributed multi-agent applications. According to the multi-agent systems approach, an application based on the JADE platform is composed of a set of cooperating agents, which can communicate with each other through message exchange. Each agent is immersed within an environment that can be acted upon and from whom events can be perceived. Intelligence, initiative, information, resources and control can be fully distributed on mobile terminals as well as on computers in the fixed network. The environment can evolve dynamically and agents appear and disappear in the system according to the needs and the requirements of the applications.

JADE provides the basic services necessary for distributed peer-to-peer applications in the fixed and mobile environment allowing each agent to dynamically discover others and to communicate with them. From the application point of view, each agent is identified by a unique name and provides a set of services. It can register and modify its services and/or search for agents providing given services, it can control its life cycle and, in particular, communicate with all other peer agents by exchanging asynchronous messages, a widely accepted communication model for distributed and loosely-coupled

software components. Communication between the agents, regardless of whether they are running in the wireless or in the wireline network, is completely symmetric with each agent being able to both initiate an interaction and respond to it. JADE is fully developed in Java and is based of the following driving principles:

- *Interoperability*. JADE is compliant with the FIPA specifications [2]. As a consequence, JADE agents can interoperate with other agents, provided that they comply with the same standard.
- *Portability*. JADE provides a homogeneous set of APIs that are independent from the underlying network and Java edition. More in details, the JADE run-time provides the same API for the J2EE, J2SE and J2ME environment and it has been designed and optimized for low footprint and memory requirements.
- *Ease of use and faster time-to-market*. The set of APIs has been designed with the goal of reducing the time to market for developing applications; therefore, they aim to hide the complexity of the middleware behind a simple and intuitive set of APIs.
- *Pay-as-you-go philosophy*. Programmers do not need to use all the features provided by the middleware. Features that are not used do not require programmers to know anything about them neither they add computational overhead.

After this introduction, the paper is organized as follows. Next chapter describes the high level components of the architecture. Then the evolution of distributed kernel on which the platform is based upon is analyzed and the new architecture based on a set of distributed coordinated filters is presented. Next chapter presents some of the most interesting platform-level services. Then, the split container architecture is presented and how it is suitable for mobile networks and devices, including how the platform is able to guarantee MSISDN-based[1] identification of the peers. A categorization of the application domains where the platform and this technology provides highest payoff is then presented and, finally, the Open Source Community, the JADE Board, and the future roadmap are presented.

2. JADE Architecture

JADE includes both the libraries (i.e. the Java classes) required to develop application agents and the run-time environment that provides the basic platform-level services and that must be active on the device before agents can be executed. Each instance of the JADE run-time is called *container* (since it "contains" agents). The set of all containers is called *platform* and provides a homogeneous layer that hides from agents the complexity and the diversity of the underlying tiers (hardware, operating systems, types of network, JVM) as depicted in Figure 1.

For bootstrapping and FIPA compliance purposes, one among these containers is labeled as *Main Container*; it must be the first to start up and all other containers

[1] MSISDN stands for "Mobile Subscriber ISDN Number". Basically it is the phone number of the SIM-card in the mobile phone.

register with it at bootstrap time. The Main Container has the following specific responsibilities:

- It manages the *Container Table* (i.e. the set of all the nodes that compose the distributed platform).
- It manages the *Global Agent Descriptor Table* (i.e. the set of all the agents hosted by the distributed platform, together with their current location).
- It manages the *Message Transport Protocols Table* (i.e. the set of all deployed message transport endpoints, together with their deployment location).
- It hosts the platform *Agent Management Service* (*AMS*) agent, mandated by FIPA specifications as unique white page and life-cycle management agent.
- It hosts the platform *Default Directory Facilitator* (*DF*) agent, mandated by FIPA specifications as default yellow page management agent.

All the above operations are essential for correct and FIPA-compliant platform operation. Unfortunately, this also entails that, from a fault tolerance perspective, the Main Container is a sensible part of the platform, and a single point of failure for many tasks. For this purpose, the replication service, described in the followings, allows to monitor and replicate the main container responsibilities.

Its distributed architecture allows deploying JADE platforms that run across multiple Java editions, from powerful servers to small mobile devices. The limited memory footprint, in fact, allows installing JADE on all mobile phones provided that they are Java-enabled. JADE is compatible with the J2ME CLDC/MIDP environment and it has already been extensively used on the field over GPRS network with several commercial mobile terminals. The JADE run-time memory footprint, in a MIDP1.0 environment, is around 45 KB, but can be further reduced until 20 KB using the ROMizing technique, i.e. compiling JADE together with the JVM [3]. JADE is extremely versatile and therefore, not only it fits the constraints of environments with limited resources, but it has already been integrated into complex architectures such as .NET or J2EE [4] where JADE becomes a service to execute multi-party proactive applications.

3. Platform Kernel

The platform is built on top of a distributed kernel that supports basic platform-level operations such as message delivery and agent life-cycle management. The current kernel architecture can be considered as the third generation of the kernel. The first generation of JADE had a monolithic distributed kernel and was only able to operate in the J2SE environment. The second generation had to target the whole spectrum of Java editions (from Micro-Edition to Enterprise Edition) and introduced the idea of having many platform profiles, that allowed choosing whether to include a feature or not. Though flexible enough, this second-generation kernel had a fixed feature set. Even if most features (agent mobility, event management, platform security) could be turned on or off, the kernel did not allow easy extension by including new platform-level services. In order to effectively address the more demanding requirements gathered from the first JADE-based systems, and also in order to promote an application model based on seamlessly situated agents, a third generation of JADE kernel was conceived and built.

This section outlines the resulting architecture, highlighting the design rationale backing up the major choices made.

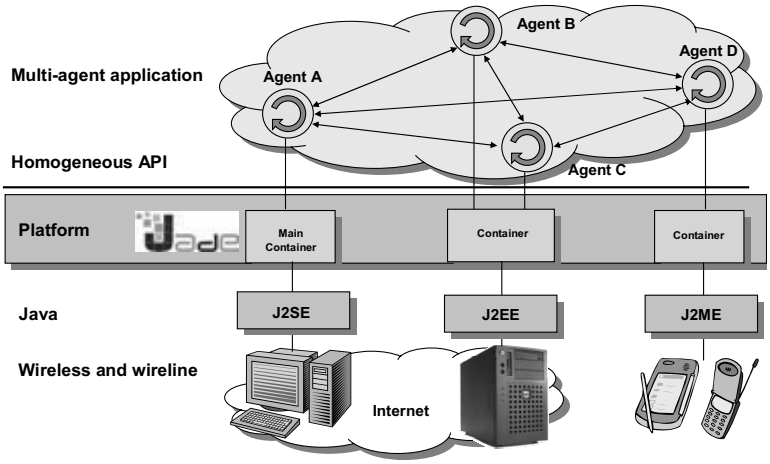

Fig. 1. JADE Distributed Architecture

3.1 Ideas and motivations

The first, natural direction in designing the third generation JADE kernel was to strive for finer-grained modularity of the kernel features. Moreover, an open-ended set of kernel-level feature sets was envisaged, and a flexible deployment strategy was essential in targeting the hybrid wireless/wireline network.

The first abstraction, adopted in order to give an extensible structure to the JADE kernel, is the *Service*. A JADE service groups together a set of features according to their conceptual cohesion, and is the kernel-level unit of deployment. Some sample JADE kernel services are agent management, agent mobility and event notification. These services closely correspond to the second-generation feature sets, but now they comply with a generic and extensible model that allows many more services to be developed and eases creating more complex ones, such as some of the services that will be described in the next sections.

The second abstraction adopted to achieve the seamless distribution of agents in the platform is the *Container*. This concept was kept from the previous kernel versions, but now it has an enhanced semantics. While in first- and second-generation kernels a JADE container was meant to contain only agents, now a third-generation container holds both agents and services.

From the previous requirements and the design vision, a set of desiderata for the new architecture was obtained:

1. Different services can be composed together.
2. Any subset of the available services can be deployed at any given container, possibly adding/removing a service during normal platform operation.
3. Some services can be present only on a subset of the platform containers or can be present with different service levels at different containers.
4. The services can run on the various Java editions supported by JADE, possibly with a graceful degradation on the more resource constrained devices.

3.2 The distributed coordinated filters architecture

Part of the inspiration needed to shape the new architecture was drawn from the research area about aspect-oriented programming [5]. The main tenet of aspect orientation is to promote separation of concerns, by writing software code as a collection of independently written aspects, expressing a different concern each.

Aspects, though written separately, will then be combined together to yield the final application. The process of combining together the different aspects according to some rules is called aspect weaving. The aspect-oriented approach basically stems from a programming language viewpoint, and the first works such as AspectJ [6] use source code translators to perform compile-time aspect weaving. Another option would be to perform run-time aspect weaving through some kind of component composition technique; among these dynamic approaches, a pioneering work was made by Aksit and others [7] with the proposal of composition filters as a way to transparently extend object systems.

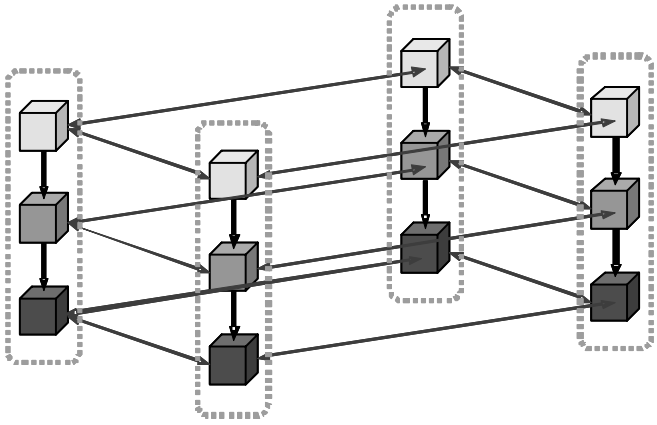

Fig. 2. Distributed coordinated filters architecture of the JADE Kernel.

In Java-based aspect languages and systems, a third option is viable and has become very popular, that is, classload-time aspect weaving. Due to the lack of support in J2ME and considering that JADE struggles to provide all its features to users through nothing more than a Java API, the third generation JADE kernel architecture took the biggest inspiration from the composition filters approach. Basically each object is provided with two filter chains: an incoming chain whose filters are invoked whenever the object receives a method call, and an outgoing chain whose filters are invoked whenever the object is about to call some other object's method itself.

By mixing the composition filters approach with the distribution of services across containers, we obtained the *Distributed Coordinated Filters* architecture, sketched in Figure 2.

In the figure above, every color refers to a specific kernel service, and every dotted line encloses a container. This means that, in the depicted example, the distributed platform has three services deployed on four containers. The various kernel operations are represented by commands, which flow across services in their container as shown by the vertical arrows. This accounts for the filtering aspect, just like in the original composition filters model. The horizontal arrows show that a single service (say, the one at the bottom) is actually distributed on several network nodes. We call *Slice* the part of a service that resides on a given node; the colored cubes in Figure 2 are exactly the service slices. Service slices and their interaction account for the distribution and coordination aspect in our model.

The UML diagram in Figure 3 shows the general layout of the resulting solution.

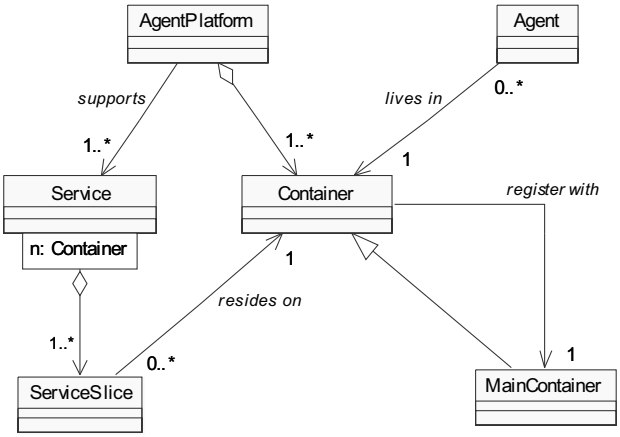

Fig. 3. Main elements of the JADE Kernel.

4. Platform Services

The architecture of the JADE kernel enables the deployment of platform-level services at run-time: each container can be deployed with the necessary services, such as communication, replication, persistence, or security. Programmers can also implement and deploy their own application-specific services, where needed.

The agent class usually accesses these platform-level services via a *Service Helper* that exposes all the methods available to the agent. In some cases (e.g. for the communication service), the methods are exposed by the Agent class itself, partly because of backward compatibility, partly because of easiness of use.

For the sake of brevity, this section does not report the full list of available services but just some of the most relevant and representatives.

4.1 Communication Service

In order to communicate, an agent just sends a message to a destination agent. Agents are identified by a name (no need for the destination object reference to send a message) and, as a consequence, there is no temporal dependency between communicating agents. The sender and the receiver could not be available at the same time. The receiver may not even exist (or not yet) or could not be directly known by the sender that can specify a property (e.g. “all agents interested in football”) as a destination. Because agents identifies each other by their name, hot change of their object reference are transparent to applications. Despite this type of communication, security is preserved, since, for applications that require it, JADE provides proper mechanisms to authenticate and verify “rights” assigned to agents. When needed, therefore, an application can verify the identity of the sender of a message and prevent actions not allowed to perform (for instance an agent may be allowed to receive messages from the agent representing the boss, but not to send messages to it). All messages exchanged between agents are carried out within an envelope including only the information required by the transport layer. That allows, among others, to encrypt the content of a message separately from the envelope.

The communication service is instantiated, by default, on every JADE container and it makes transparent to agents the location of communicating end points. Each agent owns a private queue that is filled by the communication service with incoming messages. Agents can access this private queue via any arbitrary combination of the following methods: polling-based, timeout blocking-based (i.e. blocks until a message arrives or until the given timeout expires), message-template based (i.e. it gets the first message in the queue that matches the passed message template).

The structure of a message complies with the ACL language defined by FIPA [2] and includes fields, such as variables indicating the context a message refers-to and timeout that can be waited before an answer is received, aimed at supporting complex interactions and multiple parallel conversations. As a matter of fact, in order to further support the implementation of complex conversations, JADE provides a set of skeletons of typical interaction patterns to perform specific tasks, such as negotiations, auctions

and task delegation. By using these skeletons (implemented as Java abstract classes), programmers can get rid of the burden of dealing with synchronization issues, timeouts, error conditions and, in general, all those aspects that are not strictly related to the application logic.

An analysis and a benchmark of Scalability and Performance of the JADE Message Transport System is reported in [8].

4.2 Security Service

Security in JADE is, by nature, distributed and enabled by a set of services delineated by function. Those services are *Authentication*, *Permission* and *Encryption*, described as follows:

The *Authentication Service* ensures that any user starting a JADE platform or container is legitimate within the computational system and takes responsibility for her actions, so as to be authorized to create agents within that platform. Being legitimate in the case of JADE authentication implies that the user is known to the system by having at least one valid identity and associated password. Authentication does not however imply any guarantees of behavior, this is managed by the Permission Service. The authentication mechanism itself is based on the *Java Authentication and Authorization Service* (*JAAS*) API [9] that enables enforcement of differentiated access control on system users. JAAS provides a set of de facto *LoginModules*; with the Unix, NT and Kerberos modules implemented in the current. The Unix and NT modules are Operating System dependent and are designed to use the identity of the user extracted from the current Operating System session. The Kerberos module is system independent in operation, but requires system-specific configuration prior to use. Such login modules usually require the user to enter some information as credentials, such as passwords or smart-cards. A variety of input methods have already been provided, including text, GUI and command-line, the latter available primarily for authenticating users on remote containers. The default configuration is such that if the user fails to be correctly authenticated, the system will exit and issue appropriate messages.

Due to authentication, all components (containers and agents) in a JADE platform must be owned by an authenticated user. As an extension of this, the *Permission Service* provides a layer of control over the actions that agents can perform, either permitting or denying them according to stated rules. It is thus possible to selectively grant access to platform services or application resources. This ultimately implies that permissions can be used to influence the structure of relationships between agents interacting within and across JADE platforms. The rules are typically stated in a system files according to standard JAAS policy file syntax [9], extended with a special policy model providing enhancements specifically useful to distributed agent applications. Two types of policy files can be used to grant permissions to agents: (1) The *MainContainer policy file* that specifies platform-wide permissions, e.g. "Agents owned by user Bob can kill agents owned by user Alice". (2) *Container policy files* that specify container-specific permissions, e.g. "Agents owned by user Bob can kill agents owned by user Alice on

the local container"). Container policy files can also regulate access to local resources (JVM, file system, network, etc.).

Finally, message privacy and integrity is managed by the *Encryption Service*, which provides reasonable security guarantees when sending a ACL messages between agents on the local, or a foreign, platform. Signatures are used to both ensure the identity of a message originator and the integrity of a message (confidence that data has not been tampered with during transmission). Encryption is used to ensure privacy of the message by protecting message data from eavesdropping (confidence that only the intended receiver will be able to read the clear message). In JADE both signature and encryption always apply to the entire payload of a message in order to protect all the information contained in the slots of the ACL message (content, protocol, ontology, etc.). The security-related information (such as the signature, the algorithm or the key) is placed into the envelope. Users themselves do not need to deal with the actual signature and encryption mechanisms, but just need to request a message to be signed or to check whether a received message has been signed. If some problems occur whilst signing, encrypting, verifying or decrypting a message, the message is discarded and a failure notice is returned to the sender.

4.3 Agent Management and Migration Service

The Agent Management service provides support for managing the life cycle of agents. Each agent owns and controls its thread of execution, and life-cycle transitions can only be initiated by the agent or requested to the AMS (provided that the requestor has the needed permissions). The platform takes care of hiding the object reference of the agent in order to avoid other agents, or other objects of the system, to take control and directly manipulate an agent by calling its public methods. Notice that these two features (i.e. owning the thread and keeping private the object reference) are needed in order to meet the agent autonomy requirement that each platform is requested to guarantee.

In the J2SE and Personal Java environments, JADE supports mobility of code and of execution state so that an agent can stop running on a host, migrate its code and state on a different remote host, and restart its execution from the point it was interrupted (actually, JADE implements a form of *not-so-weak* mobility because the stack and the program counter cannot be saved in Java). This functionality allows distributing computational load at runtime by moving agents to less loaded machines without any impact on the application. In a similar way, agent cloning is support in order to clone agent state and code.

4.4 Replication Service

To keep JADE fully operational even in the event of a failure of the Main-Container, a Main Replication Service was included in the latest versions of the platform. With this service is then possible to start any number of Main Container nodes, which will arrange themselves in a logical ring so that whenever one of them fails, the others will

notice and act accordingly. Ordinary containers will be able to connect to the platform through any of the active Main Container nodes; the different copies will evolve together using cross-notification. Without Main Container replication, JADE platform has a star topology, while enabling Main Container replication turns the topology into a ring of stars, as shown in Figure 4.

In the fault-tolerant configuration two or more Main Container nodes are arranged in a ring, and each node is monitoring its neighbour: if the node Main-Container-1 fails, the node Main-Container-2 will notice and inform all the other Main Container nodes, so that a smaller ring can be rebuilt with the surviving nodes.

Peripheral containers can be arbitrarily spread among the available Main Container nodes. Any single peripheral container is connected to exactly one node and in absence of failures it is completely unaware of all the other copies. When a Main Container node fails, there will generally be some orphaned peripheral containers. They will attach themselves to another one among all the Main Container nodes present in the platform.

JADE supports two policies in distributing the Main Container list to peripheral containers. A first option is to enable detection of changes to the list and notify peripheral containers. A second option is to pass the address list to peripheral containers at start-up time, avoiding notification traffic towards peripheral containers when the fixed list of Main Container nodes is known beforehand.

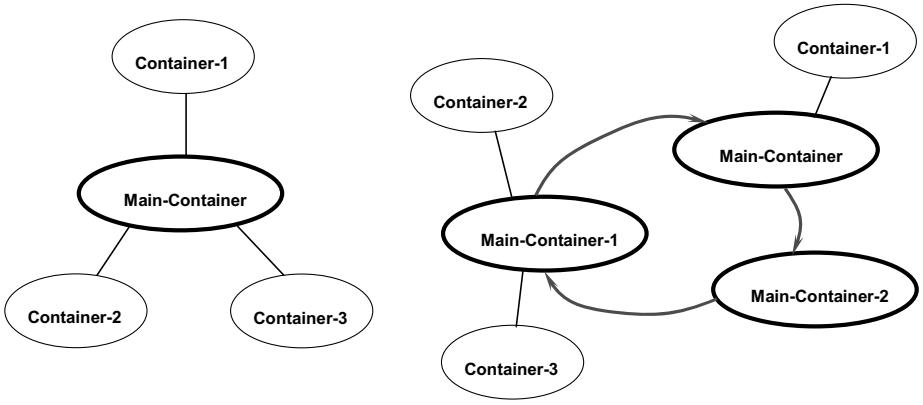

Fig. 4. Star topology (left) and ring of stars topology (right)

4.5 Message-Content Management Service

When an agent *A* communicates with another agent *B*, a certain amount of information *I* is transferred from A to B by means of an ACL message. Inside the ACL message, *I* is represented as a content expression consistent with a proper content language (e.g. SL) and encoded in a proper format (e.g. string). Both A and B have their own (possibly different) way of internally representing *I*. Taking into account that the way an agent internally represents a piece information must allow an easy handling of that piece of

information, it is quite clear that the representation used in an ACL content expression is not suitable for the inside of an agent. In order to facilitate the creation and handling of messages content, JADE provides support for automatically converting back and forth between the format suitable for content exchange (including String, sequence of bytes, XML and the *Resource Description Framework*, RDF), and the format suitable for content manipulation (i.e. Java objects). This support is integrated with *Protégé* [10], an ontology creation graphical tool that allows also importing/exporting ontology in several formats, including the *Web Ontology Language* (*OWL*).

This support for content languages and ontologies automatically performs all the necessary message format marshaling (and unmarshaling) as well as a number of semantic checks to verify that I is a well-formed piece of information, i.e. that it complies with the rules (for instance that the age of Giovanni is actually an integer value) of the ontology by means of which both A and B ascribe a proper meaning to *I*.

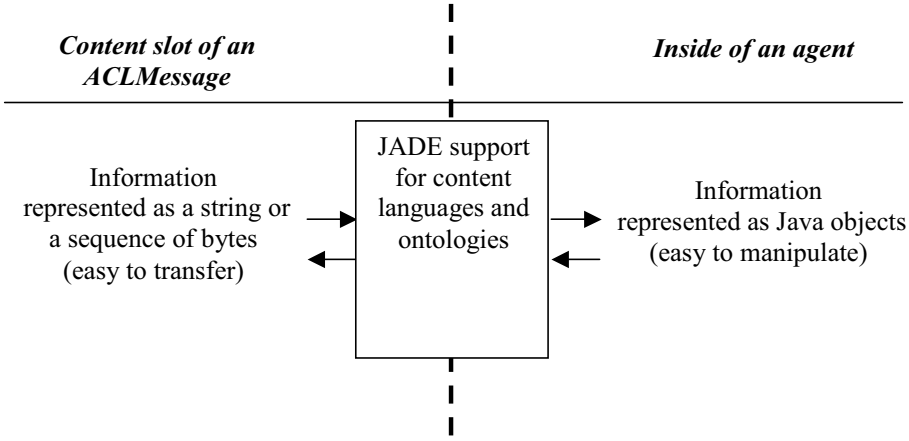

Fig. 5. JADE support for content management.

The conversion and check operations are carried out by a content manager helper object. The content manager provides a convenient interfaces to access the conversion functionality, but actually just delegates the conversion and check operations to an ontology (i.e. an instance of the `Ontology` class included in the `jade.content.onto` package) and a content language codec (i.e. an instance of the `Codec` interface included in the `jade.content.lang` package). More specifically, the ontology validates the information to be converted from the semantic point of view while the codec performs the translation into strings (or sequences of bytes) according to the syntactic rules of the related content language

Notice that JADE is opaque to the underlying inference engine system, if inferences are needed for a specific application, and it allows programmers to reuse their preferred system. It has been already integrated and tested with *Java Expert System Shell* (*JESS*

[11]) and some Prolog environments; moreover, a separate academic add-on is available, which supports building rational agents [12].

4.6 Persistence Service

Another important aspect that needs to be considered in order to support server-side applications is persistent data management. Nowadays the dominant infrastructure for this is a relational DBMS, with more recent extensions towards a less structured data model (mainly binary data and text).

The persistence-related features were divided into two categories. First, there are system-level features, such as persistently storing agents and whole containers; then, there are application-level features concerning the persistent storage of application-specific entities. In designing JADE Persistence Service, the focus for the API was kept on system-level features, with the rationale of avoiding redundant API wrapping, relying instead directly on some chosen persistence Java API for application-level features.

The chosen persistence engine turned out to be *Hibernate* [13], for a number of reasons among which its plain persistent component model and its powerful object-oriented query language were prominent. Using Hibernate data mapping capabilities, a model of the relevant JADE system-level entities (agents, containers, messages and others) was made. Then, a basic graphical tool was developed to allow easy management of persistent storage and both API-level and ACL-level access was granted to application programmers. The Persistence Service is distributed as an add-on to the main JADE package and is currently being evaluated and tried out by the JADE user community.

4.7 Other services

Several other platform-level services are available and the expandability of the kernel will facilitate several more to be implemented in the future.

For instance, a *Logging Service* is available that exploits the capabilities of the `java.util.logging` package in order to enable management of per-class logging levels and logging handlers (e.g. different file names and file formats). A graphical tool, the `LogManagerAgent`, also allows modifying at run-time the logging configuration of each class, features that is very useful for on-the-site debugging.

Of course, the platform also includes a naming service (ensuring each agent has a unique name) and a yellow pages service that can be distributed across multiple hosts. Federation graphs can be created in order to define structured domains of agent services.

5. JADE for Mobile Networks

Usage of JADE on mobile networks and resource-constrained mobile terminals needs the platform to properly address new non-functional issues, such as the memory and processing power limitations of mobile devices and the characteristics of wireless networks, specifically the commonplace *General Packet Radio Service* (*GPRS*) in terms of bandwidth, latency, intermittent connectivity and IP addresses variability. Inter-container communication cannot be anymore assumed to be stable, persistent, and reliable as connectivity may suddenly drop because of dead spots.

The *Split Container* execution mode faces these limitations by executing only a very small part of the container (called *front-end*) on the mobile terminal. The remaining part (called *back-end*) runs somewhere in the fixed network. As depicted in Figure 6, the front-end provides to agents living on the mobile phone the same APIs as a normal container: that makes the split container execution completely transparent to application agents. Similarly, a back-end behaves for the rest of the platform exactly as a normal container does: that makes the split container execution completely transparent to the rest of the platform. The front-end and the back-end are linked together by means of a permanent connection. The front-end just communicates with the back-end that, instead, is responsible for interacting with the rest of the platform. When the front-end of a split container is launched on a mobile device, it first needs to create its back-end on a host in the fixed network. In order to do that, it requires a normal JADE container to act as a mediator and to serve back-end creation requests. Any JADE container can act as mediator for several split containers, the only limitation being the number of connections that a single host is able to keep opened.

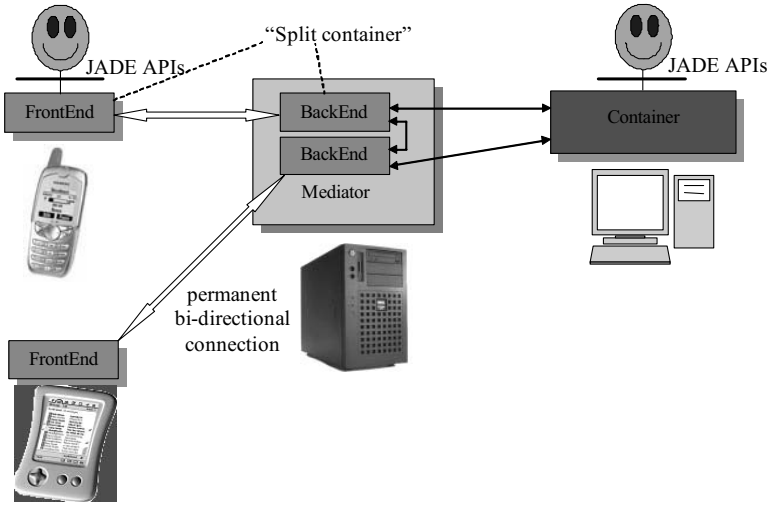

Fig. 6. Split container architecture.

The split container approach provides a number of advantages:

- Most of the container functionality (such as hosting the services and registering with the Main container) is delegated to the back-end. As a consequence, the front-end can be extremely lightweight and save resources for applications agents. Using obfuscation techniques, as described in [3], we were able to reduce the JADE footprint down to 43 Kbytes.
- The front-end is able to detect a drop down of the connection with the back-end (e.g. because the user entered a dead spot), react, and automatically recover it, as soon as possible.
- Both the front-end and the back-end can implement a store and forward mechanism so that any message, that was tried to exchange when the connection was temporarily down, is automatically buffered and delivered as soon as the connection is restored.
- The IP address of the device is hidden by the back-end to the rest of the platform. Therefore a change of the device IP address is completely transparent to the applications and to the rest of the platform.
- Last but not least, the split container approach allows the platform provider (likely the telecom operator) to keep control on the applications and the services provided over the platform. In fact, though completely peer-to-peer from a logical point of view, the communication between two agents on mobile devices always passes through the back-ends of the split containers. By plugging ad hoc services in the back-ends, the operator is in fact able to implement e.g. application-specific event-based logging, billing, priority assignment mechanisms based on the user subscription.

5.1 The PDP context manager module

Multi-agent applications need to face the peer discovery problem, i.e. how a peer becomes aware of other peers to communicate with. Typically, this is solved by means of some sort of white pages and yellow pages services that require each peer to register and successively search for other peers. Mobile devices, however, are intrinsically identified by their user phone number, i.e. the MSISDN (Mobile Station International ISDN Number). For this reason, all mobile applications involving people whose phone numbers are known a-priori (e.g. applications involving a group of people) should not require any peer discovery mechanism: a peer acting on behalf of a user should be addressable by simply specifying the phone number of that user.

JADE supports this MSISDN-based identification by means of the PDP (Packet Data Protocol) context manager module. When a mobile phone attaches to the GPRS network, it communicates with the operator's SGSN (Serving GPRS Support Node) to request the creation of a PDP context. This is a logical association between the mobile phone and the network, including aspects such as routing, QoS and billing. The mobile phone request is passed to the GGSN (Gateway GPRS Support Node) to verify the GPRS account and assign an IP address to a new PDP Context. In general, the authentication is performed by using a RADIUS server connected to the GGSN. The

JADE PDP context manager is a software module able to interact with the RADIUS Server. When a back-end creation is requested, the JADE container, which is acting as mediator, interrogates this module and gets back the MSISDN corresponding to the IP address the request comes from. In this way, JADE can use the device MSISDN to identify the starting split container and possibly force any agent starting on that container to comply with proper name space conventions based on the MSISDN. For example, an agent implementing a chat application may be named *<device msisdn>-chat* so that its name is known a priori by other chat agents on other devices. The PDP context manager module complies with a simple and well defined interface so that the actual implementation can be modified or even replaced depending on the operator network topology.

Besides offering an intrinsic peer identification mechanism, the integration with the RADIUS Server also allows exploiting the GPRS authentication to authenticate JADE nodes. The advantage of this is threefold:

- It is secure enough from the operator point of view since it relies on its standard authentication mechanism.
- It allows the operator to easily manage access control lists of users/phone-number authorized to join the platform.
- It does not require users to enter additional username and password to launch JADE based applications on their mobile devices.

6. JADE Application Domains

One of the main goals in developing a middleware is to maintain a high degree of openness and flexibility, making it applicable in as many different application domains as possible. We propose hereafter a first general categorization with no scientific completeness but just to provide a general understanding of the effectiveness of JADE. The highest payoff of adopting Agent Technology is expected in those cases where an interaction between numerous elements is required, and where an autonomous and dynamic adaptation to complex relations is needed.

6.1 Mobile applications

The focus here is supporting users on the move with a “personal agent” that helps its owner. Its goal is to facilitate the search and discovery of information through the interaction with other peers, being both other people and “service providers”. In general, JADE agents are extremely suited to act in the context of Mobile PIM – Personal Information Management: their ability of autonomous and proactive acting and seamless communications allows conceiving applications for everyday life organization, like meeting organizer, info search or services negotiation.

Several examples illustrate the strengths of JADE peers acting as assistants for travelers, being private tourists or mobile workers. Other applications have been developed in the mobile work or sales support systems, in which agent duties are task

and information sharing and exchange. In the entertainment field, JADE middleware can either be the base for multiparty gaming applications, in which a real interaction between players is offered, or can be the building block for an enhanced kind of mobile community where peer-to-peer communication allows richer relationships amongst members.

BTexact uses the JADE platform, and the LEAP add-on for mobile terminals, for its application supporting the coordination and the activities of a mobile workforce, including the distributed scheduling of jobs, job management on the fly, travel and knowledge management, and location-based coordination [14]. *Telecom Italia LAB* uses JADE for developing new mobile VAS (Value Added Services) for nomadic micro-communities over Java-enabled mobile phones.

6.2 Internet applications

In the Internet domain several application concepts based on agent technology have been proposed during the years, as the main concept has been to beneficiate from the increased connectivity, in terms of bandwidth and relations amongst people.

JADE-based systems allow end users to deal with the complexity and number of opportunities, and to exploit the possibility to seamlessly access remote resources and services. Key elements in JADE-based applications designing are direct communication support, smart information retrieval capabilities and negotiation techniques. Starting from these basic principles, various sectors have been considered such as e-learning or e-healthcare, and in general all the contexts of e-commerce/e-trading have been tackled. Moreover many of the considerations put forward in the mobile environment can been easily extended to a fixed consumption, allowing an integrated organisation of the personal and working life between PC and mobile devices. Lastly, the entertainment sector has been analysed starting from the obvious consideration on community services, multiparty gaming and content sharing applications.

Whitestein Technologies AG uses the JADE platform in the health care field, in collaboration with *Swisstransplant*, the Swiss National Transplant Coordination centre for organ transplants, for an agent-based system for decision-making support in organ transplant centers. By combining agent technology, constraint satisfaction techniques and JADE capabilities, Whitestein implemented the *Organ Transplant Management* (*OTM*) solution [15].

6.3 Corporate applications

The intelligent agents approach is an evident choice for business applications striving to enhance company productivity and efficiency. The point is easing collaboration and cooperation between systems and people in order to achieve better results. Many different examples of the usage of agent technology have been provided already in order to support company processes: when it comes the time to share information or

coordinate tasks the deployment of JADE-based system becomes very effective and useful.

Instances have been proposed in “soft contexts”, such as knowledge management and personnel administration or in general for the support to decision making processes. Other proposals have come in “harder” environments, including companies core processes, like logistic or production: valuable demonstrations concern systems for factory control, exploiting JADE capabilities in optimising tasks and coordinating resources. Interestingly enough, some JADE applications exceed even company boundaries supporting activities amongst different businesses: the ability of agents in the negotiation and information retrieval amongst services and resource providers leads to the construction of JADE-based supply-chain or e-procurement products.

6.4 Machine-to-Machine applications

In those cases where communication peers are machines instead of humans, JADE features can be exploited at its best. Autonomy and proactivity of agents play a crucial role in the management of complex systems: when the number of elements and the complexity of the relations raise, the opportunity of a distributed control significantly simplifies system operations.

Complex algorithms and heavy elaborations, typically concentrated in a single central point can be spread among agents, increasing overall efficiency and system performance and reducing the risks connected with this concentration, thus increasing system fault tolerance and scalability.

Classical examples in this context are automatic control or traffic management systems, but the helpfulness of JADE approach can go down in more depth: several studies are under way in order to extend P2P agents paradigm to network management, with the concept of peers mapping onto network equipments and interacting amongst them for resources optimizations and system control.

Finally it’s interesting to cite the extensive application of agent models for simulation uses. The concept of numerous elements acting independently but together in the environment in which they are part of, easily applies to several contexts, from different science sectors (for instance biology, ecology and natural science in general) to social and economic studies.

Rockwell Automation implemented the *Manufacturing Agent Simulation Tool* (*MAST*) [16] for the manufacturing control sector. The tool shows agents for basic material-handling components (e.g. a manufacturing cell, a conveyor belt, an AGV, etc.) capable of mutual collaboration on the product transportation via the exchange of messages. Agents are able to deal at run-time with dynamic changes and exceptional cases, such as failure detection and recovery, change of the layout of the factory’s shop floor, addition/removal of components or their interconnections.

6.5 JADE Community

The evolution strategy of the JADE project, since its beginning, is based on a collaborative intent aimed at focusing the interest of an ever-growing community of users and developers. This community revolves around two focal points: the open source project and the JADE Governing Board.

JADE is, in fact, distributed open source under the LGPL license. The aim of this license is to facilitate the creation of an open and effective platform with the help of a user community, while allowing everybody to base the business upon the applications. At the time of writing, more than 60,000 official downloads and several contributions from some tens of different companies and academic institutions have been counted, including complete subsystems (the so-called *JADE add-ons*), but much more are welcome and expected.

The JADE Governing Board [1] is a not-for-profit agreement between a set of companies, all sharing the intent and the effort of promoting the evolution of this software tool and its adoption by the mobile telco industry as a java-based de-facto standard middleware for agent-based applications. The JADE Board governs and implements the evolution of the software deciding which features should be added next. At the time of writing, the Board is composed of 5 members: *TILAB*, *Motorola*, *Whitestein Technologies AG*, *Profactor GmbH*, *France Telecom*. Each member of the JADE Board has the advantage of contributing and voting about priorities, technical and strategic decisions about the evolution of the JADE Project. Board members shall agree with the governing rules and commit to a minimal amount of resources for development and promotion of JADE. The Board is open to all companies and organizations with a concrete business interest in JADE and that commit to its development and promotion.

7. Conclusions and Future Roadmap

The paper presented the architecture of the JADE platform with particular details on the new kernel, realized as a set of distributed coordinated filters. This new kernel enables composition of platform-level services and great extensibility and adaptivity to the requirements of the deployment environment. The specific case of the mobile networks and terminals has been presented by showing how the split container execution mode allows meeting the peculiar requirements of that deployment environment and the PDP context manager module has been described, that enables support for MSISDN-based identification. As any healthy software infrastructure, JADE is still evolving while trying to balance the need for change with the requirement to protect its users from backward-incompatible modifications.

In order to manage the ever-increasing size and evolution of the project, a Governing Board was created as a not-for-profit agreement between companies that commit to the development and promotion of the platform. At the time of writing, the four main goals/directions of the project are the followings. The first goal is to consolidate the software platform by increasing the strength of features such as scalability, performance, robustness, security, integration with Web Services and network