UbiMedic: a Ubiquitous Computing Solution for Medical Emergencies

Giacomo Cabri[1], Francesco De Mola[1], Nicola Muratori[2], Raffaele Quitadamo[1], Franco Zambonelli[3]

[1]Dipartimento di Ingegneria dell'Informazione – Università di Modena e Reggio Emilia
41100 Modena, Italy
{cabri.giacomo, demola.francesco, quitadamo.raffaele}@unimore.it
[2]O2 Studio Ingeneri Associati di Muratori Nicola e Rossi Massimiliano
42020 Albinea, Reggio Emilia, Italy
nicola.muratori@oduestudio.it
[3]Dipartimento di Scienze e Metodi dell'Ingegneria – Università di Modena e Reggio Emilia
42100 Reggio Emilia, Italy
Zambonelli.franco@unimore.it

Abstract. This paper investigates the feasibility of employing the Software Agent technology in the highly dynamic and variable context of healthcare emergency coordination and decision-support domain. We introduce the design of an agent-based middleware tailored to the requirements of such context and propose a framework, called *UbiMedic*, for the implementation and deployment of services, like monitoring services, communications and remote medical measurements in injured people. From the analysis of the framework, we are able to identify some of the major technical requirements it should meet as well as challenges to be addressed for effective use in commercial applications. We choose software agents as the key enabling technology because they offer a single, general framework in which large-scale, distributed real-time decision-support applications can be implemented more efficiently.

1 Introduction

Software agent technology provides an attractive and important model for building large-scale distributed applications in heterogeneous computing environments. In particular, a "mobile" agent can be viewed as an autonomous program that has the ability to transport itself between the nodes of a network entirely under its own control, carrying with it the data and the execution state required to resume execution at the destination host [FVP98]. Due to their autonomous and active nature, agents offer several benefits over traditional technologies such as client-server, to enable a mechanism for context-aware decision support. As they are able to encapsulate the know-how needed to perform a given task, agents tend to be relatively easy to customize and can rapidly adapt to changing user requirements and run-time context. Some of current pervasive computing research

projects combine agenthood with context awareness [PAD04] and, through this convergence, mobile agents are capable of embedding light-weight context reasoning engine and can do preliminary processing in the original context.
The healthcare field is not only widely distributed and fragmented but it also exhibits a high degree of heterogeneity. The current lack of standards across different institutions and within the same institution prevents it from using a single software solution to support a cooperative working environment. The framework presented in this paper focuses primarily on the domain of medical emergency management, which is a very information-intensive and mission-critical one. In such scenarios, the hospital environment by default is ‘highly mobile’ with caregivers constantly on the move. In order to meet the varying information and resource needs of these personnel and yet be able to support their physical mobility requirements, agent-oriented computing seems to provide an ideal fit. The flexibility offered by this paradigm, however, comes with some major challenges that are infrequent in the client-server paradigm (e.g. the key issues of security and trust) and that we try to clearly highlight here.
In this paper, we advocate the pro-active, goal driven and autonomous nature of software agent technology as a support to cope with the highly mobile, dynamic and variable context of medical emergencies. Starting from this consideration, we propose an agent-based framework, named *UbiMedic*, as a helpful abstraction level upon the heterogeneity of medical *emergency* scenarios: our basic idea is that every entity involved in a healthcare environment (e.g. medical instruments, ambulances, doctors) should be represented by an agent installed in the platform. Consequently, application specific services (e.g. remotely driving medical devices, chatting with doctors, coordinating the overall emergency operations) are carried out instantiating proper agents and letting them interact, in a ubiquitous and context-aware fashion.
The paper opens up with a short report on the state of the art (Section 2) and then outlines in Section 3 the major features of a system for the support of medical emergencies. In Section 4, we describe the system composition, providing examples of possible applications that can benefit from the system, while in Section 5 we deeply analyze the components of our framework. Further, in Section 6 we argue that the research in this critical field is rather immature and needs more efforts to address a certain amount of new arising challenges. Section 7 concludes the paper and proposes an outlook of our future work towards the complete implementation of the UbiMedic framework.

2 State of the Art

The contemporary scenario of telemedicine applications presents several technological solutions, spread over different fields and use cases. However, most of the applications actually used in health care are often strictly confined to each particular experience, referred to specific companies and localities.

For example, let us consider the *LIFENET® System* by Medtronic Physio-Control [MEDweb], a multinational company that provides a full range of services and complementary products that form an emergency cardiac care system. Twelve-lead electrocardiograms and other vital signs are measured on the patient through a device called *Lifepak 12*, a portable multifunctional monitor and defibrillator. This information can be transmitted by a bluetooth connection toward the doctor's mobile phone or a *Lifenet EMS* (Emergency Medical System). This device, belonging to the ambulance equipment, is an electronic patient care reporting system (ePCR) implemented on a tablet PC: it can collect, elaborate and show all the data received from the Lifepak. Moreover, the same data can be sent by GSM or GPRS from the mobile phone to a receiving station (*Lifenet RS*), a PC installed in the hospital, where a specialist can give a second opinion to the doctors directly involved in facing the emergency.

This solution, as well as the other commercial products, has the heavy limitation of being completely closed and protected by patent rights, so that the system cannot communicate and integrate with other solutions. The limited possibility to access several services through a single integrated system is often underlined by staff in charge of medical and territorial emergencies. This implies also efficiency problems in resource employment and integration, because devices previously in use in a hospital can hardly be integrated with a new system made by a different company.

Moreover, the information technology is prevalently applied in telemedicine among hospitals and fixed health care centres, with the implementation of tele/videoconference and data transmission systems to ask for second opinions and to follow patients and treatments at a distance. Instead, the external scenario of medical and territorial emergencies still need deeper exploration, having to cope with a much more variable and complex context.

We can find interesting solutions in the field of eldercare [BellBCM05], such as the six-month study conduced at the Honeywell Laboratories to implement the *Independent LifeStyle Assistant* (ILSA) [HaiK04]. This is an agent-based monitoring and supporting system to help elderly people to live more independently at home, by reducing caregivers load. It consists of a multi-agent system supporting continuous data monitoring via home-installed sensors. The collected data are processed to obtain response planning and machine learning. In particular, ILSA is implemented over the JADE agent platform. The project development met serious difficulties due to the use of agents: the agent paradigm turned out to be too expensive for a system presenting many centralized features, so that a simpler design would have been sufficient and more convenient.

On the other hand, if we consider the CodeBlue [ShnCLFW05] experience, it seems closer to the emergency field, where distributed features are much more marked: CodeBlue is a combined hardware and software platform for medical sensor networks developed in a prototypal version at the Harvard University. Sensor networks consist of small, low-power and low-cost devices with limited computational and wireless communication capabilities. CodeBlue comprises a suite of protocols and services that let many types of devices coordinate their activities. Once again agents appeared unsuitable because of the extremely limited resources of the devices, where also other traditional approaches (like RPC and JVM) are not feasible for the same reason. Furthermore, the medical devices used to test CodeBlue, like a pulse oximeter and a two-lead electrocardiogram monitor, were developed ad hoc for the system.

3 A System to Support Medical Emergencies

As we noticed in the previous section, the field of medical emergencies is much more dynamic and subject to context variability, compared to other telemedicine fields. It often presents difficulties in communicating and even in physically reaching the place where first aid is needed. The efficiency of the assistance is an aspect of vital importance, so that all involved people (from healthcare workers to drivers) need to communicate, coordinate and access distributed resources in a very simple and fast way. The scene of the accident is not a priori known and the employed resources (users and medical devices) must dynamically organize themselves in a temporary net, where communication links are established just in time. The technology to set up the communication links must be chosen in real time depending on the particular context. Therefore, the major requirements of such a distributed system are *dynamicity* and *context-awareness*, i.e. they have to easily self-configure and adapt to the situation variability. The new system must support the emergency staff to remotely monitor and coordinate the distributed resources in real time, allowing to successfully control critical situations and patients' health. In order to make the users as accustomed and familiar as possible to the system, portability is another important requirement to take into account, so that each operator can seamlessly access the system from her preferred machine. In addition, the integration of different devices and applications through open-source technologies and common standards brings further benefits: thanks to integration, users can access different services from the same device; distinct systems can communicate and interact more smoothly; the existing equipment of the healthcare service can interoperate with new systems and devices without headaches due to proprietary solutions and incompatibilities; efforts and results coming from different experiences can be unified and integrated to obtain a more and more complete system. Integration must be intended also in communications, combining multimedial means and different channels: on the one hand, users must have the possibility to choose the best communication mode (textual, vocal, video or combined) according to their needs; on the other hand, the system must automatically choose the most efficient communication technology among the available ones at runtime. Reliability and tolerance to human errors is, in fact, an indispensable requirement in such a delicate field as emergency healthcare.

Many peculiar issues in the medical domain, in particular those belonging to the emergency field, can be addressed very well with the conceptual and methodological tools provided by the agent technology [Nea03]. Several reasons can motivate such a choice:

- each component of a multi-agent system can run on a different machine, fitting very well distributed scenarios, such as the emergency assistance one;

- thanks to their *sociality*, agents can interact, coordinate and cooperate to reach common objectives, reflecting what the different emergency units must actually do in their work;

- medical domain problems are intrinsically complex: multi-agent systems adopt decomposition techniques to tackle complex problems, partitioning the problem space into smaller tasks; this helps the system's designer focusing on some simpler portions of the system, deferring global more complex decisions to the last design stages;

- *proactivity* is another feature of software agents: a proactive agent is able to select useful actions depending on the perceived context and the explicit user intervention; this can facilitate the course of operations when maximum efficiency is required;

- the agent paradigm provides a model in which autonomous entities (i.e. agents) are assigned high-level goals to achieve and they have the reasoning capability required to take decisions by their own: they interact and mutually cooperate, keeping at the same time a certain degree of independence related to internal organization, information ownership and responsibility level, fitting very well the different institutions involved during an emergency (e.g. hospital, fire brigade, civil defence,…);

- finally, agent mobility could be beneficial in external dynamic contexts, such as those of emergencies and accidents, where the location is an unpredictable variable.

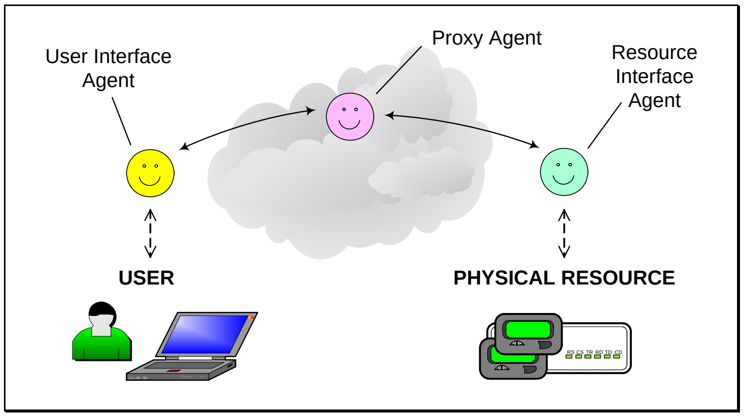

Fig. 1. Main agent typologies

The adoption of the multi-agent paradigm, however, comes at the cost of an increased development effort. On the one hand, the agent paradigm provides a methodology to deal with the domain complexity using high-level abstractions, but on the other hand, the agent adoption itself can make the development process more expensive. Other experiences have already highlighted these difficulties [HaiK04], but we argue that an appropriate architecture is mandatory to tackle medical emergency issues. In this direction, we propose a useful middleware that provides a whole set of facilities for users and application services. This will separate the system core facilities from the application services, with consistent benefits in the development process.

Such a middleware, called *UbiMedic*, is a research work aimed at building a distributed context-aware platform with mobile agents implementing helpful services for user applications. The middleware is an agent-based framework too, providing low level services to let the applications run even on small devices (e.g. PDAs or other limited user's terminals). The fundamental idea of each application accessing a medical device (e.g. an electrocardiograph or a pulse oximetry sensor) is the definition of three kinds of agents (Fig. 1): a *User Interface Agent* (UIA) responsible for managing user interactions, by means of a more or less complex GUI; a *Physical Resource Interface Agent* (PRIA), which has to collect and make available, to requesters all over the platform, the patient's vital signs measured by medical devices; a *Proxy Agent* (PA) which includes the proper logic to process and filter the collected data according to the specific source/device and to the particular goals of the application. Each time a user, through her interface, wants to interact with a medical device, a proxy agent is generated and acts as a mediator between the two entities (i.e. the UIA and the PRIA). Having a PA associated with the UIA relieves the user's device (maybe with limited resources) of the burden of executing the bulk of the code, letting the system choose the best location where to move and execute the PA, considering the amount and type of needed resources; this way, we can achieve increased portability of user's applications on a wider range of computational devices. The proposed approach does not exclude that certain applications, whose needed interactions are only among users (e.g. chat, on-map location, agenda), can be developed without the intervention of any proxy.

4 General software architecture

The proposed system relies on a middleware built upon the operating system, whose services are available to several kinds of applications of the upper level (see Fig. 2). The middleware layer offers both low and high level services (discussed later), to provide dynamicity and context-awareness features to the system.

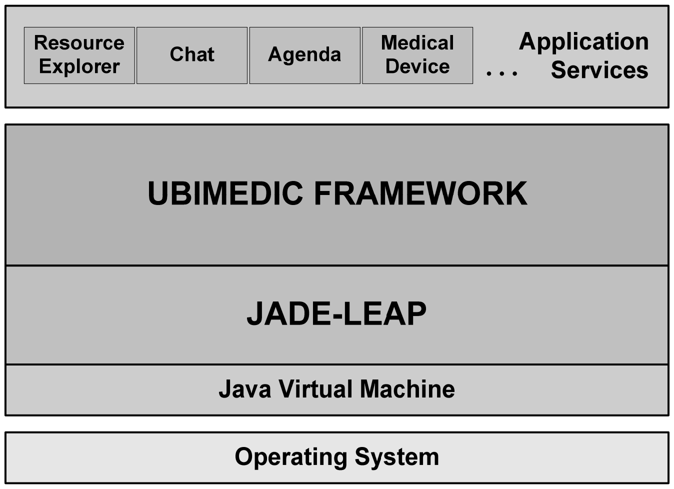

Fig. 2. A layered view of the architecture

4.1 Application Layer

The possible applications that can benefit from the services in the UbiMedic framework spread over a wide range of medical emergencies support tools: from a resource explorer, to textual and vocal chat, medical device interactions, on-map location, shared agenda, data storage, logging, administration tools, and so on. Application services are designed and implemented separately from each others: they can be installed as independent modules, progressively enriching the system with new available functionalities.

An application service carrying out interaction among users is usually composed by three entities (Fig. 3): the *Service* class, which exposes a public shared interface for a generic distributed application, like the ones listed above; an *application specific class*, extending the Service superclass, which contains the code implementing the particular functionalities of the application; a *Service Agent*, univocally associated to each instantiated service, which is the entity representing the instance of the application in the agent platform; the Service Agent is responsible for invoking the service methods of the application, acting as a mediator between the application classes and the rest of the multi-agent system, in particular the User Interface Agents shown in Fig. 1.

If the application involves the interaction with a medical device, a fourth part must be implemented instead of the described ones: a *Proxy Agent*, acting as a mediator between the User Interface Agent and the physical device. It encapsulates device specific logic, wrapped around a standard interface, in order to offer uniformity in the interaction with heterogeneous medical devices (the agents shown in Fig. 1 including the Proxy Agent are considered part of the box named "System" in Fig. 3).

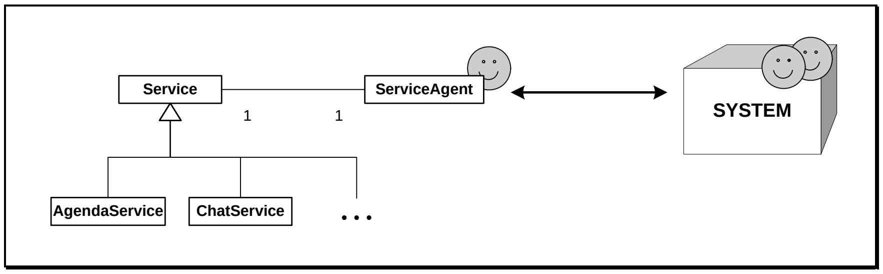

Fig. 3. The main classes of the applications

In practice, if a developer wants to add a new applicative service, he has just to implement a class inheriting from *Service*, specifying its own logic, or as an alternative a specific Proxy Agent. As mentioned above, the methods to interface the application class with the Service Agent are exposed as a standard interface by the Service class. As a consequence, each application is implemented as a plain Java application, ideally without any knowledge of the agent paradigm underneath. The advantage of this approach is in the fact that all those tricky issues, related to the used agent technology, are completely transparent to the user.
Some essential application services are:

- *Resource Explorer*: each new identity connecting to the system will always start this service first. The application provides a complete graphical view of all the connected resources (users and devices). The view is personal, according to user's profile and visibility permissions, and can be organized in trees grouping the identities on the base of their typology. The Resource Explorer, through the mediation of the Request Manager (see the next section), must interrogate principally the Discovery service (see later) to collect information about connected resources.
- *Chat*: a real-time communication channel among connected users. It can be textual, vocal or video, to meet all coordination needs during an emergency. It can be started selecting the receiver from the list shown by the Resource Explorer.
- *Medical Device Interactions*: the remote interaction with a medical device consists in visualizing its measured values and, if possible, in sending some requests to drive the device. The activation of such a service requires the instantiation of a Proxy Agent, specific of the selected medical device (the proxy code could be supposed to be provided by the device manufacturer), which will be the mediator between the physical device and the User Application Interface.

4.2 The Middleware

The middleware is composed of three main layers, described in the following. We chose to exploit existent and standard technologies to reduce the implementation work, to grant more flexibility and openness.
Just a layer upon the operating system, we have the *Java Virtual Machine (JVM)* as the common execution environment to deploy the same application on different systems and platforms. It is the lowest layer of the middleware and helps achieving isolation from the underlying hardware/OS. There are different JVMs that can be installed and configured to run on resource-constrained devices (such as the Java 2 Micro Edition), granting a good degree of portability to the system.
A *multi-agent platform* provides all the facilities to administrate a multi-agent system, ensuring the needed supports for agent's life-cycle, their interactions, coordination and mobility. As already said, the agent paradigm can be the best fit to model a complex system in a dynamic and distributed environment, like healthcare and medical emergency scenarios. The chosen platform, JADE [JADEweb], is a FIPA-compliant framework, Java-based, providing some useful services to implement inter-agent communication, node UDP monitoring, security and fault-tolerance features. The LEAP distribution enables FIPA agents to execute also on lightweight devices, such as mobile phones or PDAs running the JVM.
The proposed *UbiMedic framework* is a layer of the middleware implementing context-awareness over the JADE multi-agent platform. It is composed of several modules, grouped into two main levels: *low level services, which* are responsible for the execution environment monitoring and the interactions with it; *high level services* implement more sophisticated and evolved functionalities, directly used by the application layer. The UbiMedic framework is detailed in the next section.

5 UbiMedic Framework

UbiMedic is an agent-based middleware, implemented on top of JADE. Fig. 4 shows the low and high level services composing the framework and detailed in the following subsections

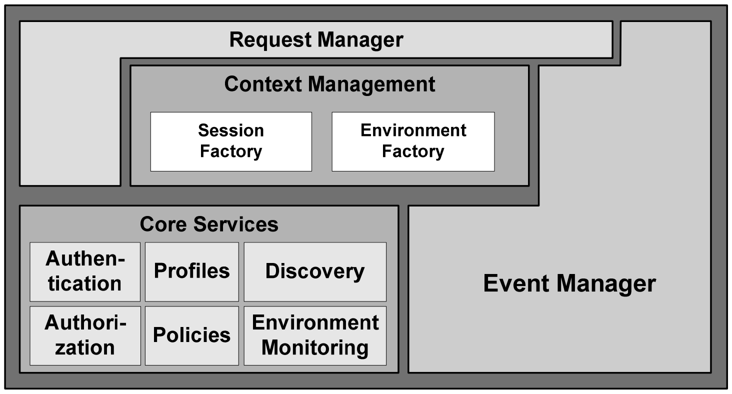

Fig. 4. The UbiMedic framework

5.1 Low level services

Low level facilities include six modules representing the core services of the system and another vertical service to collect all the possible context events. We will start the module description from the core services, followed by the Event Manager.

- *Authentication*: this module verifies credentials of the identities trying to access the system. By "identity" we mean not only each password-enabled user, but also all the active resources (e.g. medical devices) which have to connect to the system at start-up. Since a user can access the system through different devices, he must each time manually specify his credentials; the medical device credentials are automatically exposed instead by the corresponding Physical Resource Interface Agent, properly preconfigured. The Authentication module allows new connection requests if the requesting identity has the right credentials, which are maintained in a database, associated to each identity.
- *Authorization*: it is the facility ruling all the actions that identities can perform after joining the platform. Permissions and rules are stored in a database according the RBAC model [RBACweb], where multiple roles can be assigned to each identity, and multiple permissions can then be assigned to each role. The defined permissions regulate each possible action, from resource access to agent social life. No action can be performed without the acknowledgement of the Authorization module. Permissions can be expressed using declarative languages, as Ponder [PONDERweb] or Drools [DROOLSweb], and adopting the appropriate parsers and interpreters to integrate these languages in the system.
- *Profiles*: we call "profiles" all the generic parameters defining some identities' preferences, such as user's customization and device working parameters. The

corresponding core service, wrapped in the "profiles" module, retrieves these fields once again from a database and updates them, when any parameters of preferences are modified.

- *Policies*: with the term "policy" we intend preconfigured operations an identity must perform in response to some trigger events happened in the environment. This module allows the association of these actions to each identity in the platform, in a similar way as seen for permissions and profiles.
- *Discovery*: this module monitors in real time the available resources and the identities present in the system. Its aim is to show a continuously updated vision of the environment where the identities live, limited to the consciousness of the identities' presence and roles. It acts as a mediator between the connected identities, which have to register in and deregister from the discovery module, and the below platform directory and yellow pages services. The Discovery module is able to automatically perceive changes in the connected identities and to start searching agents according to the identity's requirements and the registered features.
- *Environment Monitoring*: this is a monitoring service of all the context features about the physical execution environments where each identity is running. The physical environment can dynamically change depending on the device localization, its computational capabilities, net access point and the happening of some external events. While the previous modules are mainly about context session information, the Environment Monitoring deals with the physical aspects of the context. Together they constitute the primary information-collecting phase, at the base of context-awareness implementation.
- *Event Manager*: this module collects all the context change notifications, received from the application level and the other middleware modules, and dispatches them to the interested objects. The objects that want to be informed about some context changes have to register to the Event Manager, specifying the event type they are interested in. When a middleware module perceives a context change or one running application causes a functional modification in the system, all they have to do is simply to notify the event to the same centralized Event Manager. This will deliver the received event to the high level services or applications previously registered for that particular event type. The Event Manager, differently from the other modules, can be considered a vertical service because of its ability to interact with both low and high middleware layers and application levels. To prevent the disadvantages related to the centralization of this service (such as bottleneck and a lot of traffic only for the notification transmission) the system can be configured with a federation of Event Managers, each one competent in a limited locality, achieving better system scalability.

5.2 High level services

High level facilities can be divided into *context management* and *request management*.

- Co*ntext Management*: this layer manages context information of each identity in the system and defines the data structures to keep this information updated. It is composed of two modules:
    - *Session Factory*: this module generates the necessary objects to let identities access the system. By session information we mean all those roles, permissions, profiles and policies related to an identity. Whenever an user or a resource establishes a connection to access the system, the Session Factory collects all session information provided by core services (in particular Authentication, Authorization, Profiles and Policies modules) and assigns them to a particular agent, called Context Agent, univocally associated with the new session of the identity. From this moment, the identity is represented in the system by its Context Agent, containing all the context information initially provided by the Session Factory and kept updated through the interaction with the Event Manager.
    - *Environment Factory*: this module collects all physical context information perceived by the low level service Environment Monitoring and generates a context object related to the particular location where an identity is running. This context information is location-dependent (each node has its own associated context object): if the identity moves to different locations, or the surrounding environment changes, the Environment Factory supplies the substitution of old information with the new one.
- *Request Manager*: this layer receives and tries to satisfy the identities' requests to access resources and functionalities. The identities have always to contact the Request Manager before performing any action in the system. Each request is analysed and executed according to the context state: the action is performed only if both physical context (e.g. availability of memory and bandwidth) and session information (e.g. permissions and profiles) are consistent with its execution. The Request Manager works in tight relation with the Context Management layer to know when to perform the received request. In positive case the requested resources or the execution results are returned to the initial identity.

The UbiMedic framework has been designed for medical purposes, but it is suitable to a wide range of different applications: it can be used in any field requiring a pervasive, dynamic and context-aware system.

6 Future Challenges

Some practical issues have still to be covered and some problems have to be solved before UbiMedic or a similar middleware could really be used in healthcare commercial applications. Some issues are strictly related to the adopted agent technology [Vigna04], while others are more general and inherent to the real world of the application field [Nea03].

6.1 Security

First of all, security is a very important matter: it is a fundamental issue, concerning both information confidentiality and authorized operations. Many of medical information are subject to privacy protection; they often cannot be made public to other people except those directly involved. On the other hand, the requested tasks during a medical and territorial emergency are very delicate and imply high responsibilities. The middleware technology must grant a secure implementation for accessing the information and avoiding unauthorized actions. While inter-agent communications can be encrypted to save confidentiality, the authentication of mobile and autonomous agents could be more difficult. Furthermore, both the agents and the nodes where they are running have to protect themselves, the former against malicious nodes, and the latter against malicious agents. Both the entities, in fact, could potentially try to attack respectively the node to exploit or damage its resources and the agent to take out its secrets. New technological solutions would be very useful in security issues, but on the other hand a possible way to partially face the problem could be to communicate each sensible piece of information between the agent and a central secure repository node, so that the agent carries no secret information during the migration.

6.2 Communication Standards and Common Ontologies

A system able to integrate different sources and applications has to exploit open and shared protocols. On the other hand, medical devices and other applications should as well be compliant with these protocols so as to be integrated into the system. But both devices and existing medical applications are often closed and proprietary solutions. At present, shared communication standards and complete common medical ontologies, indispensable to achieve effective integration and portability, are not used or do not exist.

6.3 Legal conformity

All healthcare activities are strictly regulated by laws and professional rules, which can differ on the base of local, national and international regulations. Besides, specific rules are defined for electronically conducted activities in substitution of traditional non-technological procedures. In particular, a multi-agent system, under an agent form, can represent official organizations and legal entities cooperating to reach some common goals. All these individuals have to respect some laws and deontological rules, which have to be reflected in the multi-agent system. Finally, many other regulations will be formalized in the future, with the adoption and diffusion of such systems.

6.4 Social and professional acceptance

Doctors and other health-care professionals run often into difficulties when new technological solutions are proposed to support their work. It can be hard for people used to work in a traditional or simply different way to adapt themselves to new methodologies, especially if they have no much time to care about these technology aspects. For these reasons, a system supporting medical emergencies must be very practical and user-friendly. But even if this requirement is satisfied, such a pervasive system has to face users' suspiciousness. In fact, both professionals and patients could not just feel safe entrusting their work and confidential information to new technologies.

7 Conclusions and Future Work

In this paper, we have discussed some of the major challenges arising from the application of the software agent paradigm to the healthcare environment, with a focus on medical emergency situations. We found that this paradigm can sometimes reveal complex and hard to exploit, but it is capable of offering, nevertheless, powerful features that make it perhaps the ideal fit in this context.
Therefore, we propose the adoption of a middleware, built on top of the JADE platform, in which medical devices, doctors and ambulances interact by means of their representative agents running and interacting in the distributed platform. Unlike traditional paradigms, agents exhibit the property of being autonomous and interactive and, coupled with mobility, they are capable of performing dynamic and intelligent inference tasks during their execution. Utilizing a framework based on the software agent paradigm, we can achieve a higher degree of flexibility by allowing applications to dynamically adapt to the changing demands of their execution environments. Application services are, instead, unaware of the agent platform and can be easily deployed as plain Java code, without having to manage all the complex details of agenthood.
The system development started with the implementation of the lowest services of the UbiMedic framework; they still need to be further detailed in the future, together with a more complete analysis of the other middleware modules. The whole framework will be tested step by step to validate the architectural choices and to identify possible single points of failure to cope with.

References

[BellBCM05] P. Bellavista, D. Bottazzi, A. Corradi, R. Montanari: "Challenges, Opportunities and Solutions for Ubiquitous Eldercare", DEIS University of Bologna Technical Report, October 2005

[DROOLSweb] Drools, Open Source project by Bob McWhirter, hosted at The Codehaus http://www.drools.org

[FVP98] A. Fuggetta, G. P. Picco, G. Vigna: "Understanding Code Mobility, IEEE Transactions on Software Engineering", Vol 24, 1998

[HaiK04] K.Z. Haigh, L.M. Kiff, L.M. Kiff, J. Myers, V. Guralnik, C.W. Geib, J. Phelps, T. Wagner: "The Independent LifeStyle Assistant (I.L.S.A.): AI Lessons Learned", in The Sixteenth Innovative Applications of Artificial Intelligence Conference (IAAI-04), San Jose, CA., Pages 852-857, 2004

[JADEweb] JADE, Java Agents Development Framework, TILAB, Torino, http://jade.tilab.com

[MEDweb] Medtronic Physio-Control, http://www.medtronic-ers.com

[Nea03] J. Nealon, A. Moreno: "Agent-Based Applications in Health Care", EU-LAT e-Health 2003, Cuernavaca, Mexico, December 2003

[PAD04] A. Padovitz, S.W. Loke, A. Zaslavsky, B. Burg: "Towards a General Approach for Reasoning about Context, Situations and Uncertainty in Ubiquitous Sensing: Putting Geometrical Intuitions to Work", 2nd International Symposium on Ubiquitous Computing Systems (UCS'04), Tokyo, Japan, 2004

[PONDERweb] PONDER, Policy Language for Distributed Systems Management Policy, Research Group, Department of Computing, Imperial College, London, http://www-dse.doc.ic.ac.uk/Research/policies/ponder.shtml

[RBACweb] RBAC, Role-Based Access Control, NIST, National Institute of Standards and Technology, http://csrc.nist.gov/rbac

[ShnCLFW05] V. Shnayder, B. Chen, K. Lorincz, T.R.F. Fulford-Jones, M. Welsh: "Sensor Networks for Medical Care", Harvard University Technical Report TR-08-05, April 2005.

[Vigna04] G. Vigna: "Mobile Agents: Ten Reasons For Failure", in the Proceedings of the 2004 IEEE International Conference on Mobile Data Management (MDM’04), Berkeley, California, USA January 2004