Journal of Computer Sciences and Applications, 2013, Vol. 1, No. 1, 5-13
Available online at http://pubs.sciepub.com/jcsa/1/1/2

DOI:10.12691/jcsa-1-1-2

Developing Pervasive and Adaptive Applications with MAADE

Enrico Franchi, Agostino Poggi, Michele Tomaiuolo*

Department of Information Engineering, University of Parma, Italy
*Corresponding author: michele.tomaiuolo@unipr.it

Received December 21, 2012; Revised January 21, 2013; Accepted February 28, 2013

Abstract Pervasive computing is one of the most active research fields because it promises the creation of environments where computing and communication devices are effectively integrated with users so that applications can provide largely invisible support for tasks performed by users. This paper presents an environment for software development, called MAADE (Multi Adaptive Agent Development Environment), and aimed at the implementation of multi-agent systems for pervasive and adaptive applications using both: *(i)* agents and multi-agent systems properties and *(ii)* composition filters for driving and dynamically adapting the behaviour of the system. MAADE allows the realization of both intentional and reactive agents, which live in an environment constituted by passive objects exposing their features in the form of available services. Thanks to its modular nature, the framework is proving particularly effective for allowing graduate students to experiment with different models and protocols used in Distributed Systems. Moreover, it is being used for the realization of prototypal applications, including an ubiquitous social networking platform, with dynamical location and proximity groups.

Keywords: *multi-agent systems, composition filters, adaptive systems, pervasive systems, online social networking*

1. Introduction

The necessity to manage and design heterogeneous systems is becoming increasingly evident in modern software development. The importance of smoothly evolving software is generally acknowledged, together with the need to support legacy software and different software modules in general, possibly developed separately and according to different paradigms. This tendency towards heterogeneous systems is made even more actual by the importance of pervasive computing and situated software systems.

Pervasive computing, in particular, is one of the most active research fields, because it promises the creation of environments where computing and communication devices are harmoniously integrated with users, so that applications can provide largely invisible support for tasks performed by users [50,51].

Several works discussed the features that make a software infrastructure suitable for the development of such environments; see, for example, [26,28,34,47,57]. The list of such features is very long and includes: adaptation, context awareness, distribution, interoperability, invisibility, mobility and scalability.

These features are not mutually independent and, in particular, adaptation can be considered a mandatory requirement for the development of pervasive applications that exhibit the previous features. In fact, adaptation allows to overcome the intrinsically dynamic nature of pervasive environments where users, devices and software components can dynamically enter, leave or move and where users can at any time require support for new tasks [3,13,21,27,46,53,54,55].

To cope with these issues various software architectures and models can be used. Proactive agents, goal-oriented behaviours and planning in general allow fulfilling user needs even in unforeseen conditions. On the other hand, not all software components necessarily benefit from cognitive capabilities, neither from being modelled this way. In real-world systems, some software agents may only need to react to environmental stimuli, and still provide the useful feature of spatial and/or social situatedness. Finally, the agent environment itself is made of passive objects, which expose their features in the form of available services. These services often rely on simplistic request-response synchronous protocols, but in general may vary from SOAP to RESTful interfaces, and may compose themselves in more complex services, on the basis of orchestration or choreography protocols.

This paper presents an environment for software development, called MAADE (Multi Adaptive Agent Development Environment), aimed at the implementation of multi-agent systems for pervasive and adaptive applications using both: *(i)* agents and multi-agent systems properties and *(ii)* composition filters for driving and dynamically adapting the behaviour of the system.

In the next sections, we briefly review agent theories and models that are relevant for MAADE; then, we introduce MAADE itself and we also discuss its features in the context of pervasive computing; afterwards, we introduce UBA, a ubiquitous social networking platform we designed over MAADE; eventually, we sketch some implementation notes and discuss about the experimentation and the future research lines.

2. Agent Theories and Models

The definition of "software agent" covers a wide variety of computer programs, ranging from so-called "heavyweight agents", with higher internal complexity and cognitive capabilities, to so-called "lightweight agents", with simple or no internal state and exposing mainly reactive behaviour. Between the two extremes there is a continuous spectrum of different types of software agents, with some features taken by both main types and: *(i)* deriving from different agent theories, for the description of the mental states of agents; *(ii)* implementing different agent architectures, which guide the development of real systems from abstract theories, and *(iii)* featuring different programming constructs or specialized languages.

Heavyweights, intentional agents. According to various cognitive theories, software agents can be usefully described in terms of the intentional stance, i.e., on the basis of their mental attitudes, such as knowledge, beliefs, wants etc. The mental attitudes could be about facts in general, but in some cases they could also regard other agent attitudes, thus allowing for higher order reasoning. There is not a single theory for describing the cognitive model of software agents. Traditional AI studies led to various mental models that have been refined and applied to software agents. One of the most widely accepted theories describes agents in terms of their Beliefs, Desires and Intentions (for so-called BDI agents) [48]. Mainly during the 1980s, there has been a shift of researchers' attention from monolithic intelligent systems to Distributed Artificial Intelligence. Wooldridge [59], in particular, developed some logic theories for describing Multi-Agent Systems. In those systems, communication plays a fundamental role. The speech act theory, deriving from previous studies presented in [2,14,15,52], has often been applied to Multi-Agent Systems, because it allows to associate message performatives and message contents with the mental state of agents. According to this theory, agent communications are essentially pragmatic actions, performed with the intention of procuring some change in their world. Among the various types of speech acts, probably the most easily distinguished are directives, e.g., for request messages, and representatives, e.g., for inform messages. Various frameworks for BDI agents have been realized, including Jason [7], JACK [58] and Jadex [45].

Lightweight, swarming agents. Lightweight agents are developed essentially in contrast with traditional AI and cognitive theories in general. The principal question was the applicability of cognitive theories to concrete problems, with highly uncertain and changing perceptions about the world, and multifaceted data, which easily become unmanageable through a cognitive approach, as reasoning typically requires an exponential complexity against the size of the problem. The theory behind the lightweight approach to develop multi-agent systems derives from various research works and in particular from the famous and provocative "subsumption architecture" proposed by Brooks [8]. In lightweight multi-agent systems, in fact, intelligence is not supposed to exist in the symbolic reasoning of agents, but it emerges as the result of the continuous perceptions and reactions of multiple simple agents, possibly coordinating their actions without explicit and direct exchange of knowledge. Swarming agents are often used for modelling and simulation [9,23]. Available implementations include Ascape [37], NetLogo [33], MASON [30], Repast [35] and Swarm [31].

Intermediate agents. Between those two contrasting theories, various practical systems are modelled using software agents that are neither fully intentional entities with cognitive capabilities, neither purely reactive entities, without any internal state. Among these systems, many are designed in terms of a Finite State Machine. The internal status of agents determines the kind of behaviours it exposes [11]. The single behaviors can range in complexity from symbolic representation and reasoning systems to much simpler stimulus-response rules. Among these systems, MANA [29] and EINSTein [22], behave on the basis of matrix multiplications, with a vector of weights representing the dynamical "personality", i.e. the effect of environmental stimuli on agents. Task Frame [10] is based on a simpler model, with an internal FSM and transitions triggered by particular events. On the other hand, more complex systems [38,39]are based on uncertain knowledge. Those propositions are represented as the nodes of a Bayesian network and processing consists in propagating evidence across the network; thus, they combine symbolic and numeric processing. Also in Agentcities, a large international research project advancing agent technologies, many deployed agents exploited the semantics of FIPA ACL for high-level service composition, yet they based their internal operation on the intrinsic FSM model of JADE behaviours [44]. Being other parts of the project developed according to a BDI model, the overall system was quite variegated, with respect to both agent technologies and models, but with all parts adhering to FIPA specifications for interoperability.

3. MAADE

MAADE is a software framework created to simplify the realization of distributed, pervasive and adaptive applications, merging the client-server and the autonomous agent paradigms. MAADE offers both a set of abstract and concrete agents.

This software framework allows the realization of systems based on two types of processes: *(i)* agents, either cognitive or not, and *(ii)* servers. An agent is an active process that can possibly have a proactive behaviour and so can start the execution of some tasks without the request of other processes. Servers are the passive entities realizing the interfaces exposed by the environment objects. They are only able to perform tasks on request of agents. Yet, they may compose, if necessary, the services offered by other servers through synchronous messages. Agents may have their own thread of execution, or rely on a thread-pool. In fact, they are generated by an extensible abstract-factory mechanism and managed by the runtime system through a terse interface. They perform tasks interacting, if necessary, with other agents and servers through synchronous and asynchronous messages. Moreover, while both servers and agents may directly take advantage of the services provided by other kinds of applications, servers are the best suited components for providing services to external applications, exposing one or more public interfaces.

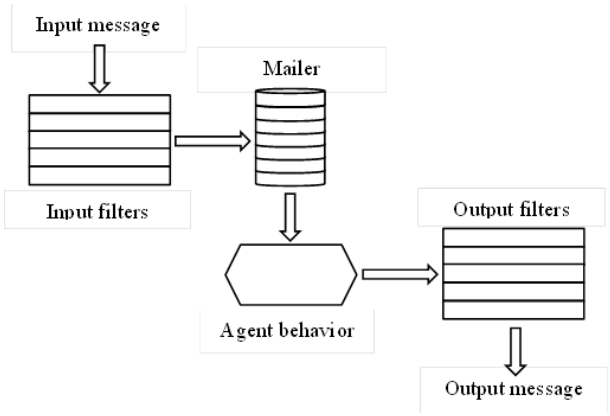

Figure 1. Simplified representation of the agent model

Servers also act as internal interfaces to application objects and in general they represent the environment sensed and acted upon by agents, either reactive or intentional. Apart from interacting with the platform environment, through available services, agents can communicate using various agent languages, as we will explain in the following sections.

Agents and servers can be distributed on a (heterogeneous) network of computational nodes (from now called runtime nodes) for the realization of different kinds of application. In particular, agents and servers are grouped into some runtime nodes that realize a platform. An application can be obtained by combining some pre-existent applications by realizing a federation.

3.1 Agent Model

An agent is an active (i.e., it has its own thread of execution) computational unit that takes advantages of six main elements: behaviour, description, description selector, mailer, message content and message filter.

An agent is able to perform one or more tasks, encoded in a behaviour taking, if necessary, advantage of the tasks provided by other agents. To facilitate the cooperation among agents, agents can advertise themselves making their description available to the other agents. The agent identifier and the agent type represent the default information contained in a description; however, agents may introduce some additional information in their description.

An agent can interact with the other agents through the exchange of messages based on one of the following three types of communication:

- synchronous communication, the agent sends a message to another agent and waits for its answer;
- asynchronous communication, the agent sends a message to another agent, performs some actions and then waits for its answer;
- one-way communication, the agent sends a message to another agent, but it does not wait for an answer.

An agent has also the ability of discovering other agents of the application. In fact, it can both get the identifiers of the other mailers of the systems and check if an identifier is bound to another mailer of the system, taking advantage of the registry service provided by MAADE runtime libraries.

Moreover, an agent can take advantage of some special objects, called description selectors, for requiring the listing of specific subsets of mailer identifiers. In fact, a description selector allows the definition of some constraints on the information maintained by the agent descriptions (e.g., the agent must be of a specific type, the agent identifier must have a specific prefix and the agent must be located in a specific runtime node) and the registry service is able to apply their constraints on the information of the registered descriptions for building the required subsets of identifiers.

An agent does not exchange directly messages with the other agents, but delegates this duty to a mailer. In fact, a mailer provides a complete management of the messages of an agent: it receives messages from the mailers of the other agents, maintains them up to the agent requests theirs processing and, finally, sends messages to the mailers of the other agents.

A message contains the typical information used for exchanging data on the net, i.e., some fields representing the header information, and an object, called content, that contains the data to be exchanged.

Normally, a mailer can communicate with all the other mailers and the sending task does not involve any operation that is not related to deliver messages to the destination; however, the presence of message filters can modify the normal delivery of messages.

A message filter is a composition filter [4] whose primary scope is to define the constraints on the reception/sending of messages; however, it can also be used for manipulating messages (e.g., their encryption and decryption) and for the implementation of replication and logging services.

Each mailer has two lists of message filters: the ones of the first list, called input message filters, are applied to the input messages and the others, called output message filters, are applied to the output messages (Figure 1 shows the flow of the messages from the input message filters to the output message filters). When a new message arrives or must be sent, the message filters of the appropriate list are applied to it in sequence until a message filter fails; therefore, such a message is stored in the input queue or is sent only if all the message filters have success.

```
ArrayList<Selector<Message>> l = new ArrayList<Selector<Message>>();

l.add(new IsInstanceOf<Message>(TEMPERATURE, new ContentGetter()));
l.add(new Equals<Message>(controller, new SenderGetter()));

Selector<Message> s = new Or<Message>(l);

Message m = take(new Or<Message>(l));
```

Figure 2. Example of use of message filters for getting specific input messages

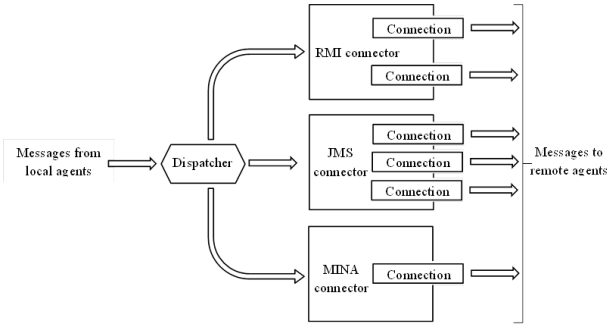

Figure 3. Distributor architecture

Message filters are not only used for customizing the reception and the sending of messages, but are also used

by the agents for asking their mailer for the input messages they need for completing their current task. In fact, as described above, a message filter allows to define the constraints that are necessary to identify a specific message and a mailer is able to use it for selecting the first message in the input queue that satisfies its constraints (e.g., the reply to a message sent by the process, a message sent by a specific agent and a message with a specific kind of content). For example, Figure 2 shows the code that an agent use for getting either a message whose content is a temperature value or a message whose sender is the controller of the application.

3.2 Agent Communication

In the previous subsection, we discussed how agents interact through messages and support three kinds of communication (synchronous, asynchronous and one-way); messages are composed of a header (containing the typical information used for exchanging data on the net), and a content object. Therefore, a message can be considered as the envelope of a traditional agent communication language (ACL) and its content can represent a performative.

In particular, MAADE does not impose a specific ACL, but allows the use of one of the ACLs provides by the environment. The current release provides: *(i)* an implementation of the FIPA ACL [18], *(ii)* a simple language, called MACL, which supports the typical request-response interaction, *(iii)* the possibility to implement another well-known ACL (e.g., KQML) [17] and *(iv)* the possibility to define new ACLs.

This solution allows the implementation of both "standard" multi-agent systems based, for example, on FIPA ACL, which allows the interoperability with other multi-agent systems, and "specialized" multi-agent systems where a "custom" ACL can, for example, reduce the development cost and/or improve the performances.

3.3 Runtime Services

The runtime libraries implement the basic services necessary for the agents of a MAADE application. These services are provided by the following four components: registry, factory, filterer and distributor.

A registry maintains the information for localizing the agents of a computational node (hereafter simply called node) of an application and allows the discovery of all the agents of an application. In fact, a registry supports: *(i)* the binding of agent identifiers of the local node with some special objects, called agent references, *(ii)* the listing of the identifiers of both the local and remote nodes and *(iii)* the retrieval of agent references, on the basis of the agent identifiers.

A reference is a proxy of the mailer of an agent that makes the communication transparent with respect to the location of the agent. Therefore, when an agent wants to send a message to another agent, it must obtain the reference to the other agent and then use it for sending the message.

A factory has the duty of creating new agents in the local node. Of course, an important side effect of the creation of an agent is the creation of the related mailer and agent reference. The creation is performed on the basis of the qualified name of the class implementing the agent and a list of initialization parameters.

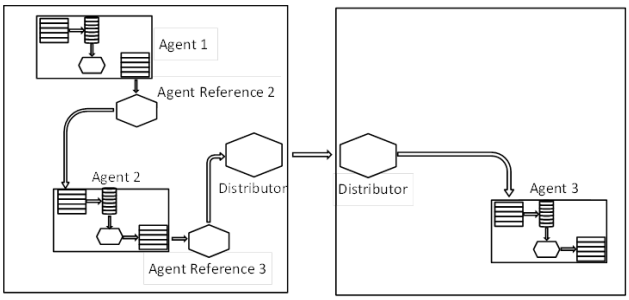

Figure 4. Exchange of messages in a distributed system

As introduced above, an agent mailer has two lists of message filters. The agents cannot directly modify such lists of message filters, but they can use a component called filterer for that purpose. A filterer allows the creation and modification of the lists of message filters associated with the agents of the local runtime node. Therefore, an agent can use such a service for managing the lists of its message filters, but also for modifying the lists of message filters associated with the other agents of the local node.

A distributor allows the communication of a node with the other nodes of an application possibly through different types of communication supports, guaranteeing a transparent communication between the agents of different nodes. A distributor has the duty of managing the connections with the distributors of the other nodes of the application. This distributor manages connections that can be implemented with different kinds of communication technology through the use of different connectors (see Figure 3). Moreover, a pair of nodes can be connected through different connections. A connector is a connection handler that manages the connections of a node with a specific communication technology allowing the exchange of messages between the agents of the accessible nodes that support such a communication technology.

A connection is a unidirectional communication channel that provides the communication between the agents of two nodes through the use of remote references (Figure 4 shows an interaction between two agents running on different nodes). In particular, a connection provides a remote lookup service that allows to the local registry both the listing of the remote agents and the access to their remote references.

3.4 Implementation

The MAADE software environment has been developed using the Java programming language. The software can be divided in an application module and a runtime module. The application module provides a set of interfaces, abstract and concrete classes that can be used for the development of systems. In particular, it contains: *(i)* a set of concrete agent and behaviour classes usable for performing the most typical tasks of the information management domain, *(ii)* a set of description selector classes usable for the most frequent discovery tasks, *(iii)* a set of message filter classes usable both for configuring security and logging services and for delegation and distribution of tasks, *(iv)* the content message classes used in the basic interactions among agents and an

implementation of the FIPA ACL, and *(v)* the artefacts (i.e., Java classes and/or configuration files) for the deployment of the nodes of a system and for the startup of the initial sets of agents and message filters. The runtime module contains an implementation of the software components that provide the runtime service described above. In particular, the distribution service has been implemented through Java RMI [41], JMS [32] and MINA [1] communication technologies.

4. Pervasive Adaptation

Providing adaptation in a pervasive environment is a challenging task as the adaptation concern affects multiple elements (devices, services, etc.) in the environment. The problem is further complicated by the fact that the elements are geographically distributed and in many instances there is no central node controlling the operation of the pervasive environment [20,48] .

MAADE integrates multi-agent systems and aspect-oriented techniques for the development of adaptive and pervasive applications. In fact, multi-agent systems are based on autonomous software entities that can interact with their environment, and therefore they adapt well to the dynamic nature of pervasive applications [12,16,19,38, 42,52,56]. Aspect-oriented techniques [14,25,49] are considered a suitable means for the development of complex applications that are composed of different interleaved cross-cutting concerns (properties or areas of interest such as energy consumption, fault tolerance, and security) and then they are indicated for providing adaptation in pervasive environments given that adaptation largely affects the other features (concerns) of such environments [20,48] .

In principle, the use of agent-oriented techniques may be sufficient for realizing effective adaptive and pervasive applications by dynamically creating new agents and by modifying the behaviour of some pre-existing agents. However, in practice, in a large set of cases an application can be adapted to the evolution of the environment without the cost of changing the behaviour of agents or of creating new agents by simply working on the delivery of messages (e.g., a particular type of message needs to be managed by another agent or the messages exchange between two agents must be encrypted). In these cases, it is necessary to have one or more control agents that drive the adaptation of an application by modifying the sets of input and output message filters associated with the agents involved in the tasks of the application.

Message filters allow the adaptation of the behaviour of an application with a limited computational overhead. In fact, each message filter usually implements a very simple task (e.g., it forwards or transforms a message) and their management is very simple (i.e., a message filter can either propagate a message to the next filter of the list or stop the propagation of the message). In particular, besides being used for implementing the typical services of an aspect oriented support, i.e., security, persistence and logging, message filters can be used for adapting the application to any hardware and software modification and for personalizing the application to the users that are using it (e.g., an input and the corresponding output message filter can adapt the interface either between two agents or between an agent and the external world, a set of messages filters, distributed on a net of agents, can route a request towards the most appropriate agent, or user, that can serve the request or can combine the tasks of different agents to provide new kinds of service).

However, although message filters can be considered the suitable “bricks” for adapting an application, they need some additional software, i.e., the agents to support evolving filter configuration. Therefore, message filters are only used to guarantee the appropriate behaviour of a pervasive application in a specific state of the environment and delegate the dynamic adaptation of the application to a set of agents. The use of agents for the dynamic adaptation of an application may introduce important overhead on the performance of the application when the reconfiguration of the message filters requires complex reasoning and negotiation tasks. However, this should be an exception and not the norm because the state of a pervasive environment evolves through a sequence of changes that rarely cause important modifications in the environment and thus the reconfiguration overhead is usually limited.

5. UBA

Online social networks have forever changed the way human being interact and socialize. Agent-based technologies are well-suited for online social networks [4], especially considering: *(i)* the networks massive scale and *(ii)* the interoperability requirements that are becoming increasingly relevant, especially in the context of pervasive computing. We decided to use the MAADE framework to implement a pervasive layer built on top of an OpenSocial Container. The system, called for simplicity UBA, was created to provide a real-world scenario to evaluate MAADE for developing pervasive adaptive applications. Moreover, UBA has an additional P2P layer that can be used for the devices to share resources without relying on the OpenSocial Container and it uses Attribute-Based Encryption [6] in order to make resources (especially on the P2P layer) and communications accessible only to those having the appropriate permissions.

OpenSocial [36] is a public specification that defines a component hosting environment (container) and a set of common application programming interfaces (APIs) for web-based applications and was originally developed to support interoperability among social networking platforms: essentially, the container holds the user's data (profile, relations and activities) that can be queried and modified by client applications. Our application does not assume any specific OpenSocial Container, however, for the deploy we used the popular Shindig implementation.

From an architectural point of view, UBA is a heavily distributed agent-based application. Each user in the system, identified with a user-id is the owner of a UBA domain, that is constituted by several UBA nodes and, eventually, each node is formed by multiple MAADE agents. UBA nodes run on specific hardware devices (e.g., a mobile phone, a tablet or a desktop computer) and every time such devices are connected to the network the corresponding UBA node is included in the user’s UBA domain. Each UBA node has a specific device-id.

However, the link between the user-id and the device-id is not public and can be disclosed only upon user's approval.

Users in the social network can be linked with multiple kinds of relationships. These relationships are expressed as belonging to a group. So, for example, @friends is a group that denotes the friendship relationship. These groups essentially work like Google Plus circles and are: *(i)* unique to a user, i.e., two groups with the same name created by two different users are two entirely different entities, and *(ii)* associated with other users or activities, e.g., creating posts or images. When a user adds other users to a given group, those users can access the resources associated with the group. If they are removed from the group, they are no longer allowed to access the associated resources.

In UBA we also have two additional kinds of groups: *(i)* Proximity groups and *(ii)* Location groups. Proximity groups are centred on each member of the social networking system and represents physical closeness to such member. Proximity groups are extremely fluid, in the sense that users can physically move and consequently the set of users belonging to a Proximity group varies in time. Each user configures the sticky-ness of his Proximity group, i.e., how long the other users are considered part of it after they are no longer physically close to him. Although a Proximity group may be entirely public, for privacy reasons it is safer to consider only Proximity groups that are subset of other groups (or to the set union of all groups, i.e., only "friends" are part of a Proximity group).

On the other hand, a Location group *(i)* is associated with the users in the proximity of a given location (e.g., a classroom or a museum room), *(ii)* has a host, i.e., a node that both identifies and supports the group and *(iii)* is associated with a location profile, which can be either hosted on the central server or on the device itself. In fact, a location, although logically different from a regular user, works in the same way and a Location group is essentially a Proximity group for the location.

As for the functionality, Proximity and Location groups mostly are similar to regular groups. However, there are some additional possibilities. When a user wants to add a resource to his Proximity group he can decide: *(i)* how to host it, *(ii)* the criteria which relate that resource to the Proximity group and *(iii)* for how long to host it. Regarding the resource hosting, the user can simply send the resource to the centralised server and host it like any other resource in a social network (e.g., a post, a picture). However, he can also decide: *(i)* to host the resource himself and *(ii)* to publish it P2P, signed for authenticity. Since the members of a Proximity group vary in time, the user must decide if the resource is be accessible *(i)* to the users which were part of the Proximity group at the moment the resource was published, i.e., the OpenSocial Agent creates a unique "anonymous" group with the users and associates the resource to that group, or *(ii)* at the moment when they access it. It is worth noticing that resources published with P2P are typically publicly accessible forever (until the resource is available) and removing them from the network is typically not feasible. However, the resource is typically encrypted.

Location groups work similarly, however, the host is typically responsible to host the resources (unless it wants to use the centralised server). P2P is still an option, especially for large media files. The main difference is the policies regarding group memberships. The set of all the Locations in the system is partially ordered (i.e., a Location can be declared to be contained by another location, but not every two locations can or should be related). The responsible organisation can provide its nodes with a semantic description of the location where the nodes are situated and their role and the nodes negotiate their order.

Location hosts determine both the time a user should be still considered a member of the group after he physically left the location and whether the user entering another location group that is not included should automatically remove the user from the present Location group. Privacy concerned users may entirely disable Location group membership and are consequently never reported to be in a given location. They can also decide never to communicate their location to specific group hosts or to whitelist the only group hosts they trust to know their position.

Several software agents provide the functionality required for an UBA node. The most important ones are:

- the Liaison Agent, responsible to make the various UBA nodes interoperate correctly
- the User Interface Agent (UI Agent), that sends notifications to the interface and takes input from the user
- the Neighbourhood Manager Agent (NM Agent), that provides a view of the reachable non-domain UBA nodes
- the OpenSocial Agent (OS Agent), that communicates with the OpenSocial Container
- the Trust Negotiator agent (TN Agent), that is responsible of trust-negotiations among different UBA domains
- the Resource Manager Agent, that manages resources in the P2P layer

An instance of each of these agents exists in every UBA node and it cooperates with its siblings in the other nodes. For example, if a given UBA node is not connected to the Internet but can sense using Bluetooth another UBA node in the same domain (i.e., belonging to the same user), the Liaison Agent of the offline node sets a composition filter that transparently forwards all the messages for its own OS Agent to that of the online node. The Liaison Agent also interacts with the MAADE registry so that the nodes in the same domain can communicate properly and in general is responsible to make all the different UBA nodes part of the same domain work together harmoniously.

However, the details of cooperation are mostly left out from the single agents. Most situations are simply dealt with setting the appropriate composition filters so that *(i)* all the relevant nodes have all the information they need to make the proper choices and *(ii)* messages sent to agents that are clearly not able to perform the task (for example, because they need to be online but the node is offline) are suppressed.

It is worth noting that UBA domain is constituted by multiple UBA nodes, but the user is probably watching only a subset of devices running UBA nodes. Thus, it is necessary to determine the active node, i.e., the node to which the user is actually connected. The Liaison Agents elect the Active Node *(i)* taking into account which is the device that registered an explicit user action or *(ii)* having the UI Agents ask the user to select the device he is currently using. The position of the user is determined

considering the position of the Active Node. However, the user can configure the system so that notifications are sent not only to the Active Node UI Agent, but also to other UI Agents. The forwarding of the notifications is entirely managed using composition filters, so that all the agents in a node only send messages to the same node UI Agent and the messages are subsequently modified to reflect their actual provenience and then delivered where appropriate.

Moreover, when the user performs some activity on the social network, such as writing a post or tagging a picture, a message is sent to the OS Agent in order to update the OpenSocial Container. Composition filters set by the Liaison Agents forward the message to the other nodes so that their view of the user profile is directly updated without needing to access the container.

A typical mobile device has several ways to scan its surroundings: Bluetooth, WiFi Direct, regular WiFi, Near Field Communication (NFC) and a UBA node has specific agents to discover other UBA nodes using these protocols. Then the NM Agent aggregates information from these agents, trying to present a consistent view, merging the data from the different sources and it configures the discovering agents according to high level criteria, such as battery consumption and hardware availability. Moreover, the NM Agent also determines its physical position (possibly using GPS, if available) and notifies the OpenSocial Agent (in order to update the profile, if the user so desires). However, the Liaison Agents activate composition filters that discard update position messages that do not come from the Active Device. Moreover, a NM Agent that senses other NM Agents belonging to the same domain informs the Liaison Agent that, eventually, sets composition filters so that all the messages coming from its sensor agents are forwarded to that device as well.

So, for example, if a user is sitting in front of his desktop computer, writing a post, the computer runs the Active Node. However, his mobile phone is equipped with more hardware sensing the surrounding and both the computer and the mobile device are connected to the same WiFi network. The NM Agents are mutually aware of their respective existence and, as a consequence, the UBA node on the computer is receiving data on the neighbourhood from the mobile phone, that is presented consistently to the user. Moreover, the UBA Node on the computer is aware of the geographical position because of the UBA node on the mobile phone (that has a GPS sensor) and can properly update the profile on the OpenSocial Container. The whole process is essentially transparent because the agents are mostly oblivious of the forwarding process.

Another important function of the NM Agent is to update the composition of the Proximity group with the UBA nodes it senses. However, when the agents responsible to assess the neighbourhood (e.g., using Wi-Fi Direct) discover a new node, they are only given its device identifier. This identifier alone cannot be used to understand the user identifier, i.e., the discovering node knows that another node is in its neighbourhood, but not to whom it belongs (unless it already knows that particular device). In order to resolve the device identifier to the actual user, the NM Agent asks the OS Agent to resolve the device-id using the OpenSocial Container. If the users are “friends” (or otherwise connected) the container can disclose the required user-id. Moreover, the sensed node is sent a message encrypted with the users’ private key that contains both a secret and the sensing node user-id. If the node actually belongs to a friend, then it can decrypt the message and send back a message encrypted with it with its private key containing the secret and its user-id. Whichever of the two processes is successfully completed, the two NM agents have the respective user-ids and can update their Proximity groups. Otherwise, a trust negotiation starts.

The TN Agent of the discovering node can: *(i)* start a trust negotiation with the discovered agent, disclosing some information (e.g., the information that both of them are members of the same class) or *(ii)* ask some other node he is already acquainted with (or potentially a Location host) to be a broker. In the latter case, both nodes can ask to a third agent to back their claims. If the negotiation succeeds, the only immediate results are that the discovering node: *(i)* knows the actual user-id of the discovered node; *(ii)* is in its Proximity group, which means that it is able to access only resources part of the Proximity group. However, typically during the negotiations each of the two TN Agents obtains some information on the other user and consequently decides to which additional groups to add him. This process can be automatic or semi-automatic (awaiting human confirmation) depending on the preferences.

UBA agents present different degrees of autonomy and intelligence. For example, agents such as the Wi-Fi Direct Agent are mostly reactive agents that only inform the NM Agent when a new UBA node is discovered. They are implemented as MAADE agents mainly because MAADE servers are not able to poll the device hardware without external stimuli. Similarly the UI Agent is implemented with a MAADE agent because it is important to send asynchronous messages in certain situations. Moreover, we named “agent” the Resource Manager Agent or the OS Agent only for uniformity, since they are implemented with MAADE servers and they have almost no agency. Moreover, we wanted to make clearer the distinction between the external server that is the OpenSocial Container and the OS Agent, which acts as a client for that server and a server for the other UBA agents.

On the other hand, the Liaison Agent, the NM Agent and the TM Agent require all the features provided by MAADE Agents. The Liaison Agents perform the elections and auctions necessary to determine the global behaviour of the UBA domain. The NM Agent processes semantically meaningful data coming from the sensor agents and tries to present a consistent view. Moreover, it configures the sensors in order to maintain energy consumption within parameters. Eventually, the TN Agent performs complex negotiations with the goal to discover the identity of the other nodes in the surroundings, disclosing the least possible amount of information on their user.

6. Conclusions

This paper presented a software framework, called MAADE, which has the goal of simplifying the development of pervasive applications by combining multi-agent and aspect-oriented techniques. In fact, this solution allows to couple the power of multi-agent based

solutions with the simplicity of compositional filters solutions guaranteeing both a good adaptation to the evolution of the environment and a limited overhead to the performances of the applications.

MAADE is proving to be an effective framework for the development of various kinds of applications, including *(i)* simulations and *(ii)* pervasive systems. In the last year, we used it in the lab activities of the "Distributed Systems" course of the computer engineering master degree. In particular, the students of the course used it both for implementing some coordination algorithms taught during the course and for developing an individual project as part of their course evaluation. Therefore, at the end of their work, MAADE was used for the development of about thirty different systems (e.g., some agent-bases simulators, some e-commerce systems, some information sharing systems, etc.). The big result of such experimentation is that all the students with a good knowledge of the Java language were able to autonomously develop a complete system after few hours of training (8 hours) on the use of MAADE. Moreover, we compared the works of students with similar abilities in the building of Java pervasive systems by using (or not) MAADE. The result in this case was that student usually spent more time when they do not use MAADE and the measure of the gap is the 20% of the development time. Of course, during their project we have discussions with them about the possible software engineering solutions, about the goal of their project, but very little time was spent for discussing about MAADE.

Future research activities will be dedicated, besides to continue the experimentation, improvement and validation of the MAADE software framework, to the development of the collaborative services for the social networking platform. In particular, current activities are dedicated to: *(i)* the implementation of more sophisticated adaptation services based on message filters taking advantages of the solutions presented by PCOM [3] and by PICO [27], *(ii)* the automatic creation of the Java classes representing the content of messages from OWL ontologies taking advantage of the O3L software library [43], and *(iii)* the extension of the software environment with a communication technologies to provide both high-performance communication and an easy integration with the Web, i.e., jWebSocket [24].

References

[1] Apache Foundation. *MINA software Web site*. Available: http://mina.apache.org/ [Accessed 2012-12-15].

[2] Austin, J.L. "*How to do Things with Words: The William James Lectures delivered at Harvard University in 1955*". Ed. J. O. Urmson, Oxford, Clarendon. 1962.

[3] Becker, C., Hante, M., Schiele, G., Rotheemel, K. "PCOM - A component system for pervasive computing". Proceedings of the *2nd IEEE Conference on Pervasive Computing and Communications*, PerCom 2004 (pp. 67-76). Orlando, FL.

[4] Bergenti, F., Franchi, E., Poggi, A. "Agent-based Social Networks for Enterprise Collaboration". In *Enabling Technologies: Infrastructure for Collaborative Enterprises* (WETICE), 2011 20th IEEE International Workshops on (pp. 25-28). IEEE.

[5] Bergmans, L., Aksit, M. "Composing crosscutting concerns using composition filters". *Communications of ACM*, 44(10): 51-57, 2001.

[6] Bethencourt, J., Sahai, A., Waters, B. "Ciphertext-policy attribute-based encryption". *IEEE Symposium on Security and Privacy*. (2007), 321-334.

[7] Bordini, R.H., Hübner, J.F., Wooldridge, M. "*Programming Multi-Agent Systems in AgentSpeak using Jason*". Wiley (2007), 292 pages.

[8] Brooks, R. "A robust layered control system for a mobile robot". *Technical Report AI Memo 864, Artificial Intelligence Laboratory*, Massachusetts Institute of Technology. 1985.

[9] Cabri, G., Ferrari, L., Zambonelli, F. "Role-Based Approaches for Engineering Interactions in Large-Scale Multi-agent Systems". *Software Engineering for Multi-Agent Systems II*, Lecture Notes in Computer Science 2940: 243-263, 2004.

[10] Camps-Mura, D., Pérez-Costaa, X., Sallent-Ribesb, S. "Designing energy efficient access points with Wi-Fi Direct". *Computer Networks*, 55(13), 2838-2855, 2011.

[11] Ceranowicz, A., Nielsen, P.E., Koss, F. "Behavioral Representation in JSAF". Proceedings of *Ninth Annual Computer Generated Forces and Behavior Representation Conference* (2000). Orlando, FL.

[12] Chakraborty, D., Takahashi, H., Suganuma, T., Takeda, A., Kitagata, G., Hashimoto, K., Shiratori, N. "An adaptive context aware communication system for ubiquitous environment based on overlay network". Recent Advances In Computer Engineering. Proceedings of the *12th WSEAS International Conference on Computers* (2008), 832-837. WSEAS, Stevens Point, Wisconsin.

[13] Cheng, S., Garlan, D., Schmeri, B.R., Sousa, J.P., Spitznagel, B., Steenkiste, P., Hu, N. "Software Architecture-Based Adaptation for Pervasive Systems". *Trends in Network and Pervasive Computing*, Lecture Notes In Computer Science, 2299: 67-82, 2002.

[14] Cohen, P.R., Levesque, H.J. "Rational interaction as the basis for communication". *Intentions in Communication* (1990), pp. 221-256. Cohen, Morgan, Pollack, eds., The MIT Press.

[15] Cohen, P.R., Perrault, C.R "Elements of a plan based theory of speech acts", *Cognitive Science* 3. 1979.

[16] Filman, R., Elrad, T., Clarke, S., Aksit, M. "*Aspect-Oriented Software Development*". (2004). Addison-Wesley.

[17] Finin, T., Fritzson, R., McKay, D., McEntire, R. "KQML as an agent communication language". Proceedings of the *3rd International Conference on information and Knowledge Management* (1994), 456-463. Gaithersburg, MD.

[18] FIPA Consortium. *FIPA Specifications*. Available: http://www.fipa.org/ [Accessed 2012-12-15].

[19] Fok, C., Roman, G., Lu, C. "Agilla: A mobile agent middleware for self-adaptive wireless sensor networks". *ACM Transactions on Autonomous and Adaptive Systems* 4(3), 1-26, 2009.

[20] Fuentes, L., Gamez, N., Sanchez, P. "Aspect-oriented design and implementation of context-aware pervasive applications". *Innovations in Systems and Software Engineering*, 5(1), 79-93, 2009.

[21] Funk, C., Schultheis, A., Linnhoff-Popien, C., Mitic, J., Kuhmunch, C. "Adaptation of Composite Services in Pervasive Computing Environments". Proceedings of *IEEE International Conference on Pervasive Services* (2007), 242-249. Istanbul, Turkey.

[22] Genesereth, M.R., Ketchpel, S.P. "Software Agents". *Communications of the ACM*, 37(7), 48-53, 1994.

[23] Ilachinski, A. "Artificial War: Multiagent-Based Simulation of Combat". *World Scientific* (2004), Singapore.

[24] jWebSocket Team. *jWebSocket software Web site*. Available: http://jwebsocket.org/ [Accessed 2012-12-15].

[25] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J. "Aspect-oriented programming". Proceedings of the *European Conference on Object-Oriented Programming* (ECOOP 1997), Lecture Notes in Computer Science, 1241, pp. 220-242. Aksit, Matsuoka eds. Springer-Verlag, Berlin, Germany.

[26] Kindberg, T., Fox, A. "System Software for Ubiquitous Computing". *IEEE Pervasive Computing*, 1(1), 70-81, 2002.

[27] Kumar, M., Shirazi, B.A., Das, S.K., Sung, B.Y., Levine, D., Singhal, M. "PICO: A Middleware Framework for Pervasive Computing". *IEEE Pervasive Computing*, 2(3), 72-79, 2003.

[28] Kumar, M., Zambonelli, F. "Middleware for pervasive computing". *Pervasive Mobile Computing*, 3(4), 329-331, 2007.

[29] Lauren, M.K., Stephen, R.T. "Map-Aware Non-uniform Automata (MANA) – A New Zealand Approach to Scenario Modelling". *Journal of Battlefield Technology*, 5(1, March 2002) 27ff.

Available: http://www.argospress.com/jbt/Volume5/5-1-4.htm [Accessed 2012-12-15].

[30] Luke, S., Balan, G.C., Panait, L.A., Cioffi-Revilla, C., Paus, S. "Mason: a Java Multi-Agent Simulation Library". Proceedings of *Agent 2003 Conference on Challenges in Social Simulation*.

[31] Minar, N., Burkhart, R., Langton, C., Askenazi, M. "*The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations*". Working Paper 96-06-042 (1996), Santa Fe Institute.

[32] Monson-Haefel, R., Chappell, D. *Java Message Service*. O'Reilly & Associates (2000).

[33] NetLogo Web site. Available: http://ccl.northwestern.edu/netlogo/ [Accessed 2012-12-15].

[34] Niemelä, E., Latvakoski, J. "Survey of requirements and solutions for ubiquitous software". Proceedings of the *3rd International Conference on Mobile and Ubiquitous Multimedia* (2004), 71-78. College Park, MD.

[35] North, M.J., Collier, N.T., Vos, J.R. "Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit". *ACM Transactions on Modeling and Computer Simulation*, 16 (2006), 1-25.

[36] OpenSocial specifications. Available: http://www.opensocial.org/ [Accessed 2012-12-15].

[37] Parker, M.T. "What is Ascape and Why Should You Care?" *Journal of Artificial Societies and Social Simulation*, 4(1). 2001.

[38] Parunak, H.V.D., Nielsen, P., Brueckner, S., Alonso, R. "Hybrid multi-agent systems: integrating swarming and BDI agents". Proceeding of the *4th international conference on Engineering self-organising systems*, ESOA'06. Springer-Verlag Berlin, Heidelberg, 2007.

[39] Pearl, J. *Probabilistic Reasoning in Intelligent Systems*. Morgan-Kaufmann (1988), San Francisco, CA.

[40] Pham, H., Paluska, J.M., Saif, U., Stawarz, C., Terman, C., Ward, S. A dynamic platform for runtime adaptation. Pervasive and Mobile Computing, 5(6), 676-696, 2009.

[41] Pitt, E., McNiff, K. *Java.rmi: the Remote Method Invocation Guide*. Addison-Wesley 2009.

[42] Platon, E., Mamei, M., Sabouret, N., Honiden, S., Parunak, H.V. "Mechanisms for environments in multi-agent systems: Survey and opportunities". *Autonomous Agents and Multi-Agent Systems*, 14(1), 31-47, 2007.

[43] Poggi, A. "Developing Ontology Based Applications with O3L". *WSEAS Transactions on Computers*, 8(8), 1286-1295, 2009.

[44] Poggi, A., Tomaiuolo, M., Turci, P. "Service Composition in Open Agent Societies". Proceedings of *WOA*, 92-99, 2003.

[45] Pokahr, A., Braubach, L., Lamersdorf, W. *Jadex: A BDI Reasoning Engine. Multi-Agent Programming*, 149-174, 2005.

[46] Porekar, J., Dolinar, K., Jerman-Blazic, B. "Middleware for Privacy Protection of Ambient Intelligence and Pervasive Systems". *WSEAS Transactions on Information Science & Applications*, 3(4), 633-639, 2007.

[47] Raatikainen, K., Chrisrensen, H.B., Nakajima, T. "Application requirements for middleware for mobile and pervasive systems". *ACM SIGMOBILE - Mobile Computing and Communications Review*, 6(4), 16-24, 2002.

[48] Rao, A.S., Georgeff, M.P. "Modeling rational agents within a BDI-architecture". Proceedings of *Knowledge Representation and Reasoning* (Apr. 1991), 473-484. Fikes & Sandewall eds. Morgan Kaufmann Publishers, Inc.

[49] Rashid, A., Kortuem¡ G. "Adaptation as an Aspect in Pervasive Computing". Proceedings of the *Workshop on Building Software for Pervasive Computing* at the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSA 2004. Vancouver, Canada.

[50] Saha, D., Mukherjee, A. Pervasive Computing: A Paradigm for the 21st Century. *Computer*, 36(3), 25-31, 2003.

[51] Satyanarayanam, M. "Pervasive Computing Vision and Challenges". *IEEE Personal Communications*, 6(8), 10-17, 2001.

[52] Searle, J.R. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press (1969).

[53] Sheu, R., Czajkowski, M., Hofmann, M.O., Schow, G. "Multiagent-based adaptive pervasive service architecture (MAPS)". Proceedings of the *3rd workshop on Agent-oriented software engineering challenges for ubiquitous and pervasive computing*, AUPC 2009 (pp. 3-8). New York.

[54] Sousa, J.P., Poladian, V., Garlan, D., Schmeri, B., Shaw, M. "Task-based adaptation for ubiquitous computing". *IEEE Transactions on Systems, Man, and Cybernetics*, 36(3), 328-340, 2006.

[55] Soylu, A., De Causmacker, P., Desmet, P. "Context and Adaptivity in Pervasive Computing Environments: Links with Software Engineering and Ontological Engineering". *Journal of Software*, 4(9), 992-1013, 2009.

[56] Tesauro, G., Chess, D.M., Walsh, W.E., Das, R., Segal, A., Whalley, I., Kephart, J.O., White, S.R. "A Multi-Agent Systems Approach to Autonomic Computing". Proceedings of the *3rd international Joint Conference on Autonomous Agents and Multiagent Systems* (2004), 464-471. New York, NY.

[57] Tweedale, J., Ichalkaranje, N., Sioutis, C., Jarvis, B., Consoli, A., Phillips-Wren, G. "Innovations in multi-agent systems". *Journal of Network and Computer Applications*, 3(30), 1089-1115, 2007.

[58] Winikoff, M. "Jack Intelligent Agents: An Industrial Strength Platform". *Multiagent Systems, Artificial Societies, and Simulated Organizations*, (15), Multi-Agent Programming II, 175-193, 2006.

[59] Wooldridge, M. "*The Logical Modelling of Computational Multi-Agent Systems*". PhD thesis, Department of Computation, UMIST, Manchester, UK (Oct. 1992). Also available as Technical Report MMU–DOC–94–01, Department of Computing, Manchester Metropolitan University, Chester St., Manchester, UK.