Developing Pervasive Agent-based Applications: A Comparison of two Coordination Approaches

Inmaculada Ayala[1], Mercedes Amor[1], Lidia Fuentes[1], Marco Mamei[2], and Franco Zambonelli[2]

[1] Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga, Spain
{ayala,pinilla,lff}@lcc.uma.es
[2] Dipartamento di Scienze e Metodi dell'Ingegneria
Università degli studi di Modena e Reggio Emilia, Italy
{marco.mamei,franco.zambonelli}@unimore.it

Abstract. Pervasive computing is concerned with making our lives easier through digital environments that are sensitive, adaptive, and responsive to human needs. Different works have shown the suitability of the agent paradigm for the development of pervasive applications. However, so far there is not a dominant approach adopted for the development of agent-based pervasive systems. In particular, two key classes of approaches exist, based on FIPA interaction protocols and tuple spaces. The contribution of this paper is the comparison and evaluation of tuple spaces and FIPA-compliant coordination mechanisms for the development of pervasive applications. To do so, we are going to compare two approaches that exemplifies these agent technologies: MalacaTiny-Sol and SAPERE.

Keywords: Pervasive computing, Agent Platforms, Tuple spaces, Evaluation, FIPA, Aspect Orientation

1 Introduction

Pervasive computing is about making our lives easier through digital environments that are sensitive, adaptive, and responsive to human needs [1]. Pervasive computing proposes the development of a new generation of advanced systems, in which cheap, interconnected computing devices are ubiquitous and capable of helping users in a range of tasks [2]. Different technologies are contributing to the development of this vision such as distributed computing, mobile computing, human-computer interaction, expert systems or agent technology, just to mention a few.

Different works have demonstrated the suitability of the agent paradigm for the development of pervasive applications, because of their capacity to be autonomous, reactive, proactive and social [3, 4]. In recent years, pervasive applications based on agents have become a reality, with different projects that exploit agent properties to implement adaptive applications. In these projects, agent

technologies have been adapted to these new environments composed of heterogeneous devices and communication means. Agents have been embedded in new devices such as smartphones or sensors, and agent middlewares have been extended to support the heterogeneity, adding new network wireless technologies, and new communication paradigms to facilitate the development of these applications. Agents have been used as abstractions to model and implement both functionality and devices of an Ambient Intelligence systems, to encapsulate artificial intelligence techniques, and to coordinate the different elements that compose the application.

The works presented in [3, 4] highlight that there has still not been a common approach adopted for the development of pervasive systems based on agents. In particular, one can think of two radically different models to coordinate agents that compose a pervasive application: indirect coordination based on tuple spaces and coordination based on FIPA interaction protocols. These two options have a great impact on the design of the agent and consequently, in the design of pervasive applications. In tuple-based approaches, agents interact by exchanging tuples, which are ordered collections of information items. Agents communicate, synchronize and cooperate through tuple spaces by storing, reading, and consuming tuples in an associative way [5]. In these approaches, agents have a simpler design because the tuple spaces embeds most of the logics of coordination. Contrarily, and in accordance with FIPA [6], agents are fundamental actors of a domain, which are able to provide a number of services and provide the functionality of the application and integrate in their code the coordination strategies too. In FIPA approaches, agent communication is based on message passing, where agents communicate by sending individual messages to each other, which are distributed through the Agent Platform (AP).

The contribution of this paper is a detailed comparison and evaluation of tuple spaces and FIPA-compliant approaches for the development of pervasive applications. The goal of this comparison is to illustrate the advantages and disadvantages of these approaches in the development of these applications, and where one approach is more advantageous over another. In order to do this, we are going to use two agent systems that exemplify these agent communication and coordination models: MalacaTiny-Sol [7] as an example of FIPA-based communication, and SAPERE [8] as an example of tuple-based coordination. Using both approaches, we are going to design a few case studies in an Intelligent Museum (IM) in order to compare the resulting systems. In this evaluation we are going to asses the internal design of the agent system, and in addition other important properties of pervasive systems such as adaptability, robustness or privacy. While this paper is grounded on the comparison between MalacaTiny-Sol and SAPERE, we believe most of our analysis can be extended to other FIPA based versus tuple space based implementations.

This paper is structured as follows: Section 2 presents the two approaches that are going to be used in the evaluation, MalacaTiny-Sol and SAPERE. Section 3 describes how to use both approaches to model different scenarios in an IM.

Section 4 accomplishes the comparison and evaluation of both approaches and the paper finishes with a conclusions section.

2 Background

In this section we will present MalacaTiny-Sol and SAPERE. These agent systems exemplify two radically different models to coordinate agents that compose a pervasive application: indirect interaction using tuple spaces and direct interaction using interaction protocols. The first one is a traditional coordination model which allows agents to interact uncoupling communicating agents in both time and space by allowing agents to communicate without knowing each other's identities [5]. In order to communicate in an asynchronous way agents read, consume, write or create new tuples in the shared tuple space. The rules (or laws) that govern coordination in the tuple space are defined outside the agents involved.

A large portion of the community considers interaction protocols, i.e. predetermined patterns of interactions, as a means to coordinate MAS [9]. It is the coordination model proposed by FIPA to be supported by FIPA-compliant APs. Internally, as part of their behaviour, interacting agents control that the message exchange complies the protocol rules. In order to support the social ability of interacting agents, exchanged messages include the intention of the agent (by means of the so-called performative). Unlike tuple-based coordination, the initiator agent needs to know the identity of its counterpart in the interaction. To support this feature, the AP provides an agent directory facilitator.

Table 1 summarises the main features of MalacaTiny-Sol and SAPERE, showing that they also differentiate in the distribution infrastructure. However, they have some features in common such as the agent architecture type, and some of the devices and network technologies they support.

Table 1. Overview of MalacaTiny-Sol and SAPERE.

Feature	MalacaTiny-Sol	Sapere
Agent architecture	Reactive	Reactive
Model of coordination	**FIPA**	**Tuple space**
Supported devices	J2SE-enabled, Android-enabled, **J2ME-enabled**, **Sun SPOT**, **Waspmote**	J2SE-powered, Android-powered
Network technologies	**802.11**, 802.15.1, **802.15.4**	802.15.1
Distribution Infrastructure	**Centralized**	**Distributed**

Despite the agents of both systems have reactive architectures, their internal design is quite different. The design principle of MalacaTiny is the enhancement of the internal agent architecture, by means of separating the domain specific

functionality from other concerns, mainly related with the coordination and exchange of messages. MalacaTiny agents interact according to the FIPA specifications and standards. In SAPERE, agents have a simpler design because the tuple space embeds most of logics of coordination.

Regarding the requirements of pervasive systems, these approaches focus on different issues. Principally, MalacaTiny-Sol deals with the heterogeneity of devices and communication technologies presented in many pervasive computing scenarios. While SAPERE provides a natural metaphor to develop applications that are distributed in a physical space. More details on these technologies are provided in the following subsections.

2.1 MalacaTiny and Sol

MalacaTiny-Sol is a FIPA compliant agent system, which adapts and extends standard agent technologies to facilitate the development of pervasive applications. In this system we can distinguish two parts: MalacaTiny [10], that allows to develop agents for lightweight devices; and Sol [7], which is the middleware where these agents are deployed and provides a set of (FIPA) services for those agents (i.e. the AP).

MalacaTiny is an implementation of the Malaca agent architecture [11] for lightweight devices. This agent technology is based on component and aspects[3], which promote the separation of application specific functionality from communication related concerns. In general, in Aspect Oriented (AO) approaches, crosscutting concerns are identified as those concerns that appear to be dispersed in different components of the system, usually tangled with other functionalities. These crosscutting concerns are encapsulated as independent entities named *aspects*. At compilation or runtime, the aspect behavior is again composed at specific points of the system execution described by the so-called *join points* in a process known as *weaving*.

In MalacaTiny these crosscutting concerns are identified in the context of a FIPA-compliant interaction, and are related with the specific functions or task that the agent has to perform in order to coordinate with other agents. The considered crosscutting concerns (which are then encapsulated as aspects) are: the formatting of messages (*Representation* aspect), the distribution of the messages using different communication means (*Distribution* aspect), and the coordination, both internal (*Context-awareness* aspect) and external (*Coordination* aspect) for the agent. The join points where these aspects are invoked are the reception and the sending of a message, and event throwing. Aspects are composed at runtime by an aspect weaver ruled by a set of explicit composition rules defined outside of the aspects involved.

The different versions of MalacaTiny are embedded in Android devices, mobile phones with MIDP profile, desktop computers, Sun SPOTs [12] and Libellium waspmotes [13]. MalacaTiny agents can be executed on top of different APs

[3] Aspect-Oriented Software Development `http://aosd.net/`

and using different transport protocols, by simply plugging in the correct distribution aspect. For instance, by using the Jade-Leap plug-in, MalacaTiny agents can communicate with other agents registered in this platform. However, current APs for lightweight devices are not entirely capable of managing both device and transport protocol heterogeneity, and have strong limitations for ensuring communication interoperability in pervasive systems. The Sol AP has been created to cope with these limitations.

FIPA-based agents require a set of services from the FIPA AP that are related with the transportation of messages between agents, and with the discovering of agents and services. Sol is a FIPA-compliant AP specially well suited to develop applications in the Internet of Things. This AP acts as an agent-based middleware that provides a set of services for the agents and behaves as a gateway to support communication heterogeneity. Specifically, the Sol AP supports:

- The registering and discovering of agents (Agent Management Service-AMS).
- The registering and discovering of services (Directory Facilitator-DF).
- The registration and membership of groups (Group Management Service - GMS).
- The message communication service (MTS), which allows the communication between agents registered in the AP, extended to facilitate the group-based communication.

Note that the AMS, DF and MTS are classical services provided by any AP, but the MTS is extended to support group communication in IoT environments, in conjuntion with the GMS.

Therefore, the main features of this AP (see Fig. 1) are the support for communication of agents in heterogeneous devices, coping with heterogeneous transport protocols (WiFi, Bluetooth and ZigBee) and group communication often required by pervasive systems. Additionally, Sol has remote nodes (Sol Clients in Fig. 1), which communicate with the node in which Sol is running. The development of these clients has been necessary for the implementation of applications distributed in wide areas. Sol clients support devices with low-range communication technology such as mobile phones that use Bluetooth, Sun SPOTs and Libellium waspmotes. These clients can run in desktop computers and Meshlium Xtreme routers [14].

A group is a way to identify a set of agents that are interested in the same type of information. Forming groups enables the Sol AP to implement multicast communication efficiently, which facilitates the distribution of the same information to clustered components of the system. Groups are defined attending to the communication needs of the applications, and agents join and leave these groups at runtime (by the GMS). Groups are usually composed of agents that share some feature (e.g. they are embedded in the same type of device) or play the same role in the MAS (e.g. agents that provide the same service).

In summary, the combination of MalacaTiny and Sol (MalacaTiny-Sol) is a system to deal with the requirements imposed by pervasive computing systems. MalacaTiny agents can take advantage of using the Sol AP, so that they can communicate through different transport protocols and send multicast messages to

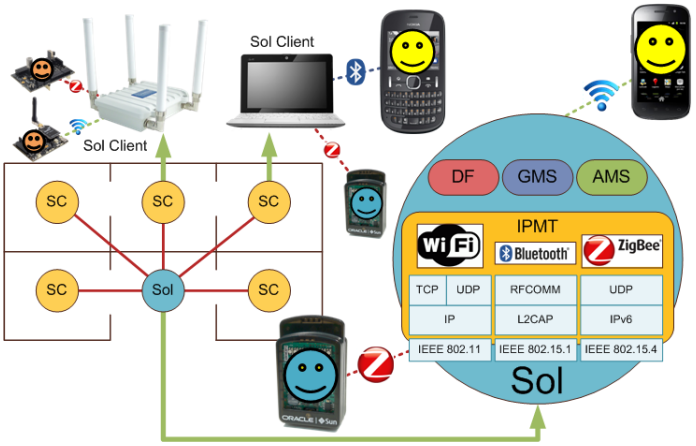

Fig. 1. Schema of the communication in Sol agent platform.

a group of related agents. With this approach, the functionality of the pervasive system is decomposed in a set of MalacaTiny cooperating agents that use the Sol AP for the location and communication between agents. Sol enables interaction via a centralized registration and discovery services. Agents communicate via message exchange and concerns are separated by aspect-based programming via the MalacaTiny framework.

2.2 The SAPERE middleware

SAPERE follows a rather different approach for the development of Multi-Agent applications. SAPERE models a pervasive service environment as a non-layered *spatial substrate*, laid above the actual pervasive network infrastructure. The substrate embeds the basic laws of nature (or *eco-laws*) that rule the activities of the system. It represents the ground on which the components of the pervasive service ecosystem interact and combine with each other. All "entities" living in the ecosystem will have an associated semantic representation: *Live Semantic Annotations* (LSAs), which is a basic ingredient for enabling dynamic unsupervised interactions between components. From an implementation point of view, SAPERE relies on lightweight and minimal middleware infrastructure (see Fig. 2). In particular, it reifies LSAs in the form of tuples, dynamically stored and updated in a system of highly-distributed tuple spaces spread over the nodes of the network [15]. Each LSA acts as an observable interface of resources and service of the components. LSAs of different components can bind with each other to enable interactions. The *eco-laws* are the rules driving the dynamics of the ecosystem. In particular, eco-laws perform pattern matching operations on the set of LSAs that are in the ecosystem to: *(i)* create bindings among LSAs, thus enabling interactions between components, *(ii)* diffuse LSAs across the spa-

tial substrate, *(iii)* aggregate LSAs together, to compute summaries of the LSA population, *(iv)* delete LSAs that are not useful.

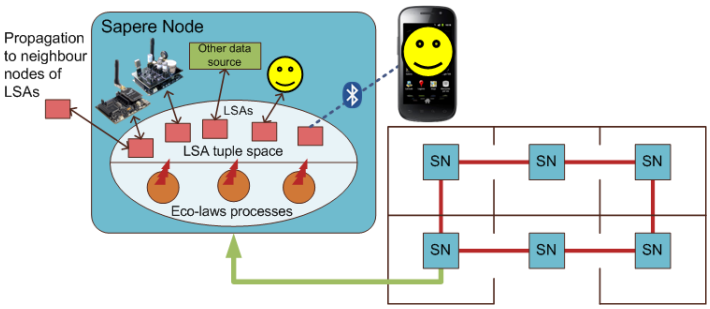

Fig. 2. The SAPERE Conceptual Architecture

The active components of the ecosystem (whether services, software agents, sensing/actuating devices, or data sources) express their existence via LSAs injected in the local tuple space associated with their node. Then, they indirectly interact with each other via the tuple space by observing and accessing their own LSA.

In SAPERE, we enforce a notable separation of concerns between application's computation and interaction. Computation (i.e., the main application business logic) is coded in the SAPERE agents using standard software engineering methodologies. Interaction consists of writing agents' LSAs and managing their evolution over time. Specifically, programmers have to specify the format of agent's LSAs so that they match with eco-laws, enabling eco-law functionalities: bonding, spreading, aggregating and decaying. In more detail, the eco-laws represent sorts of virtual chemical reactions between LSAs, and get activated by processes embedded in tuple spaces (which make SAPERE tuple spaces different to traditional tuple spaces). Such processes evaluate the potential for establishing new chemical bonds between LSAs, the need for breaking some, or the need for generating new LSAs by combining of existing ones. In addition, to support distributed spatial interactions, eco-laws can enforce the diffusion of LSAs to spatially close tuple spaces, e.g., for those tuple spaces that are neighbor of each other in the network, according to specific propagation patterns (gradient-based diffusion, broadcast, or multicast).

In summary, in SAPERE agents, interactions are mediated by the set of LSA spaces where inject LSA in the system, and subscribe to the arrival of LSAs. LSAs spread across the network enabling distributed operations.

3 Modeling pervasive scenarios

As stated in the introduction, in order to illustrate and evaluate how both approaches work in pervasive systems, we will use an IM, which put together dif-

ferent case study applications. Modern museums' buildings usually include a considerable number of displays and sensors distributed in their rooms, with the goal of providing valuable information to staff and visitors. What characterizes the IM as a pervasive system is the use of sensors and personal devices of people to enhance their experience during the visit. Moreover, the information provided by these devices can be used to improve the efficiency of the running of museum. Specifically, we are going to model scenarios of information provision (Subsection 3.1) and emergency evacuation (Subsection 3.2) in the IM.

Information provision is a very important class of applications to enrich the IM experience. In particular we will focus on: (i) monitoring of the environmental conditions of a room; (ii) controlling the number of people that are currently in the museum; (iii) and the distribution of exhibit information according to a user profile. The first two scenarios are services of interest for the security staff members, while the third is service targeted for museum visitors.

In addition, we are going to model a service that contributes to the evacuation of the building in case of an emergency. This is a a service of great importance in crowded buildings like museums. This problem can be resolved in very different ways according to the characteristics of the two approaches used. In order to illustrate the advantages and disadvantages of both, we are going to consider two situations: there is just one emergency exit; and in the case of there are more than one emergency exit.

The design of the above scenarios in MalacaTiny and SAPERE has some points in common. In both systems agents are service providers and consumers and they interact in order to provide services to the people in the museum. Other point in common is in both approaches each guard and visitor have personal agents that are running in their personal devices, and which provide them with the IM services. Additionally, both designs include an agent that represents each exhibit in the museum, and provides information about it. The last point in common is the physical distribution of the middleware because in both solutions they are distributed throughout the building. However, Sol follows the schema depicted in Fig. 1 with a main node a multiple clients and SAPERE follows the schema of Fig. 2 with multiple SAPERE nodes deployed in each room and interconnected.

The main differences are found in the type of agents considered and in the internal design of these agents. In addition to the agents previously mentioned, the MalacaTiny-Sol system incorporates agents to sensors, while the SAPERE system considers a specific agent for counting the number of visitors around a SAPERE node. Although the details of the internal design of agents for the different scenarios will be described in the following subsections, it is important to emphasize that the design of MalacaTiny agents is based on component and aspects, which specify the application functionality and interaction with other agents. So, the description of the architecture of these agents consists of describing the set of components and aspects that compose an agent, and exactly how they relate. However, agents in SAPERE have a very simple design (see Fig. 3) and their behavior emerge from the interactions with the SAPERE node. This

interaction depends on the LSAs that the agent injects into the LSAa space and the result of the application of eco-laws to these LSAs. So, the description of these agents is given in terms of injected LSAs and the behavior of the agent when these are bonded, read, removed or updated.

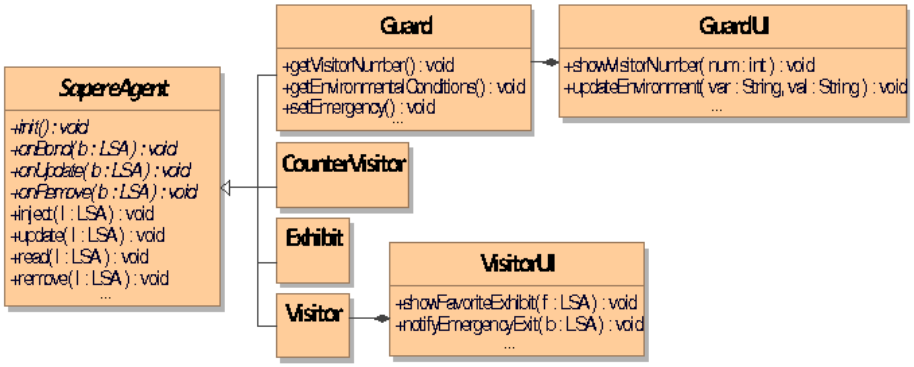

Fig. 3. UML class diagram of SAPERE agents in the Intelligent Museum

3.1 Information provision scenarios

Information provision is a very important class of applications to enrich the IM experience. In particular we will focus on: (i) monitoring of the environmental conditions of a room; (ii) controlling the number of people that are currently in the museum; (iii) and the distribution of exhibit information according to a user profile.

Designing applications with MalacaTiny-Sol In these scenarios agents interact to provide information to their corresponding users. In Malacatiny-Sol, agents exchange messages through the Sol AP. This means that the four types of agents that compose the MAS have a distribution and a representation aspect to send and receive messages using the Sol AP named *SolPlugin* and *Representation* (see Figures 4 and 5). To make the interaction between agents more efficient, we define and use groups (introduced in Subsection 2.1). As stated before, with the GMS provided by the AP we can register different groups in order to support the application requirements: one group includes all the visitor agents registered in the AP; another group comprises all the sensor agents that are deployed in a specific room (so there is a group formed for each room with sensors installed); and the last group is for the exhibits that are located in an specific room (so there is a group formed for each exhibition room).

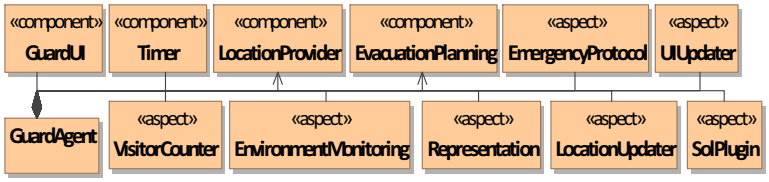

Fig. 4. UML class diagram of the agent for guards.

Environmental monitoring application. In order to monitor the environmental conditions of rooms, several sensors with agents embedded inside are deployed in them. At initiation, each agent joins the group corresponding to their room. When a security staff member wants to know the conditions in one of the rooms, his agent interacts with the group of sensor agents associated with the room, in order to gather up-to-date information and present the results to the security guard. The implementation of this scenario requires the addition of two aspects and one component (see Fig. 4) to the security guard agent: the *EnvironmentMonitoring* aspect, which requests the information from the sensor group, gathers the answers and updates the internal knowledge of the agent with it; the *UIUpdater* aspect, which updates the user interface with the new environmental results when it observes a change in the agent knowledge; and the *GuardUI* component, which implements the user interface. Components are added to the agent architecture with an identifier using the method *addComponent* (see Fig. 6) and aspects are added means of aspect composition rules. As stated before, these rules set how aspects are composed at specific points in the agent execution.

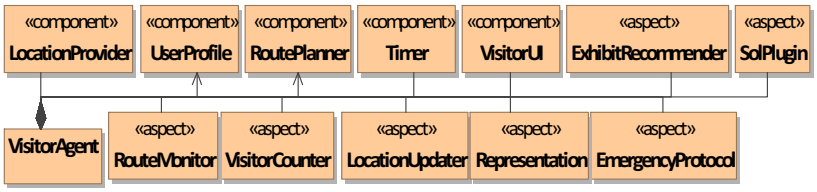

Fig. 5. UML class diagram of the agent for visitors.

Visitor counter application. In order to determine the number of visitor in the IM, the agent for guards must interact with each visitor agent. To make this interaction more efficient, again, we make use of groups. In this case, the guard agent sends an "is alive" request message to the visitor agents group previously defined, and it counts the responses over a time span. To accomplish this task, the security guard agent has to include new components and aspects in its architecture: the *VisitorCounter* coordination aspect, which collects the answers

from visitors; and *Timer* component, which determines the time span of the collection. On the other hand, the design of the agent for visitors (see Fig. 5) also includes the aspect *VisitorCounter* that joins the corresponding group at initiation, intercepts the "is alive" request and answers the request of the security agent.

In MalacaTiny, interaction protocol behaviors are implemented as finite state machines whose transitions are driven by internal events or received messages, and that cause the execution of plans. In the case of *VisitorCounter* protocol (see Fig. 7 left side), transitions are driven by events from the user interface that indicate that user requests the number of visitors (*CounterRequestEvent*), messages from visitor agents and the internal event that indicate the end of the time span (*TimerEvent*). Plans of this protocol are: *SendGroupMessage* that sends a message to the group of visitors; *ReceiveAnswer* that processes the answer from visitors and counts the number of visitors (see Fig. 7 right side); and *PresentResults* that presents the results to the security guard.

```
public class GuardAgent extends Agent{
        ....
    protected void setup(){
            ....
        addComponent("UI",guardUI);
        addComponent("Timer",new Timer());
        addComponent("GPS",new LocationProvider());
        addComponent("Evacuation",new EvacuationPlanning());
    }

    protected void compositionRules(){
        addCompositionRule(SND_MSG, Role.REPRESENTATION, ..., AuroraRepresentation.class.getName(),...);
        addCompositionRule(SND_MSG, Role.DISTRIBUTION, ...., SolPlugin.class.getName(), true,....);
                                ....
        addCompositionRule(RCV_MSG, Role.REPRESENTATION,..., AuroraRepresentation.class.getName(),....);
        addCompositionRule(RCV_MSG, Role.COORDINATION, ...., VisitorCounter.class.getName(),....);
        addCompositionRule(RCV_MSG, Role.COORDINATION, ...., EnvironmentMonitoring.class.getName(),....);
                                ....
        addCompositionRule(THRW_EVNT,Role.CONTEXT_AWARENESS,...,UIUpdater.class.getName(),....);
        addCompositionRule(THRW_EVNT,Role.CONTEXT_AWARENESS,...,LocationUpdater.class.getName(),....);
                                ....
    }
}
```

Fig. 6. Partial code of the agent for guards in MalacaTiny.

Information provision according to user profile application. This third scenario provides visitors with information about exhibits in the room where they currently are. The presented information depends on the user personal profile. This scenario requires the visitor agent to know the room where the user is, in order to interact with the agents for exhibits located in the room. The location of the visitor can be obtained internally by the agent using different mechanisms like the communication network [16]. Each time the visitor moves to another room, the agent changes the group of exhibit agents it has to request the information from. For this purpose, the visitor agent sends a message to this group and when it receives the answers from the exhibit agents, it analyzes the profile of the vis-

itor, and filters the information received to show the information of interest to him/her.

Visitor Counter Protocol

```
public class VisitorCounter extends CoordinationAspect{
        ...
    protected void setup(){
        ProtocolState initial=new ProtocolState(this,"initial");
        ProtocolState reception=new ProtocolState(this,"reception");

        InstancePattern counterRequest=
              new InstancePattern(new CounterRequestEvent());
        InstancePattern timerEvent=new InsancePattern(new TimerEvent());
        MessagePattern counterProtocolPattern=new MessagePattern();
        groupProtocolPattern.setProtocol("VisitorCounterProtocol");

        registerTransition(counterRequest, initial,
              reception,SendGroupMessage.class.getName());
        registerTransition(counterProtocolPattern, reception, reception,
              ReceiveAnswer.class.getName());
        registerTransition(timerEvent, reception, initial,
              PresentResults.class.getName());

        setInitial_state(initial);
    }
}
```

Receive answer from visitor

```
public class ReceiveAnswer{

    protected void setup(){
        ACLMessage msg=(ACLMessage)getInput();
        Integer visitorCounter=(Integer)getAgent().
              getKnowledge("visitorCounter");
        visitorCounter++;
    }
}
```

Fig. 7. Partial codes of the *VisitorCounter* protocol (left) and the *ReceiveAnswer* plan (right) in MalacaTiny.

This application is implemented in different aspects of the visitor agent (see Fig. 5): The *LocationProvider* component provides the current location of the agent, notifying a change in the user's position by throwing internal events; The *LocationUpdater* aspect takes the location information, processes it and updates the internal knowledge of the agent with it; and *ExhibitRecommender* aspect ensures that each time the user changes the location to a different room, it interacts with the exhibit agents to gather information and recommend specific exhibits to the user (according to the information in the *UserProfile* component).

Designing applications with SAPERE The communication of SAPERE nodes is based on Bluetooth and entities connect to it on a proximity basis. This means that any non mobile element of the IM, like sensors, is automatically connected to the closest SAPERE node. Additionally, in the case of agents embedded in mobile personal devices, they are continuously connecting and disconnecting nodes depending on their proximity to them.

Environmental monitoring application. Using SAPERE, sensors accomplish the environmental monitoring and provide it via the injection of LSAs in the SAPERE node that they are connected to. When a guard requests this information, his agent injects LSAs to subscribe to information about environmental conditions. When eco-laws are fired, these LSAs are bonded to he LSAs injected by sensors and the results are presented to the guard. This application is modeled differently when the security guard is not in the room from which he wants to know

the environmental conditions. To do this, it is necessary to have specific agents to gather the conditions and send the information to the remote space when is requested. This procedure is illustrated in the following scenario, when the guard agent wants to know the number of visitors in the IM.

Visitor

```
<LSA name="visitor" value="inma"/>
```

Guard

```
<LSA name="museum-visitor" value="0"/>
<LSA name="number-visitor" value="*"/>
<LSA name="number-visitor" value="*" …
spread="direct" destination="main-hall"
source="room5" .. />
<LSA name="number-visitor" value="*" …
spread="direct" destination="room3"
source="room5" ... />
...
```

```
Agent Guard {
                                    ...
    int sentLSA,recLSA,visitorCounter;

    onBond(LSA b) {
        if(b.name.equals("number-visitor"){
            visitorCounter=visitorCounter+b.value;
            recLSA++;
            updateLSA(name="museum-visitor",value=counter);
            if(recLSA==sentLSA){
                updateUI("museum-visitor",visitorCounter);
            }
        }
                                    ...
}
```

Fig. 8. Injected LSAs (left) of agents for visitors and guards and partial code *onBond()* method (right) of the agent for guard in the number of visitors scenario.

Visitor counter application. The modeling of this solution in SAPERE requires the collaboration of three types of agents: security guards, visitors and agents that count the number of people around a SAPERE node. To count all the visitors in the IM it is necessary to know the number of visitors around a SAPERE node and later, to add this information. The interaction between agents for visitors and visitor counter agents is used to determine the number of visitors around a SAPERE node. On the one hand, agents for visitors inject an LSA indicating the presence of their users around the node (see Fig. 8 left) and on the other hand, visitor counter agents are subscribed to this information and update an LSA that contains the current number of visitors around the node (see Fig. 9). These agents increase the counter when LSAs are bonded (user is in the room where the SAPERE node is deployed) and decrease it when they are removed (user leaves the room).

The process for the addition of this information starts with a request of a security guard. Then, his/her agent injects LSAs to request the information injected by visitor counter agents, to do so it has to inject an LSA for each SAPERE node with direct spreading to these nodes (see Fig. 8, left). When these LSAs arrive at their destination, they are updated with the information of the number of visitors and sent back to the node of the guard agent. In this node the agent for the guard has injected LSAs to add the values and when it receives all the answers it presents the results to the guard (see Fig. 8, right).

Information provision according to user profile application. The provision of information according to the user profile in SAPERE has an advantage over

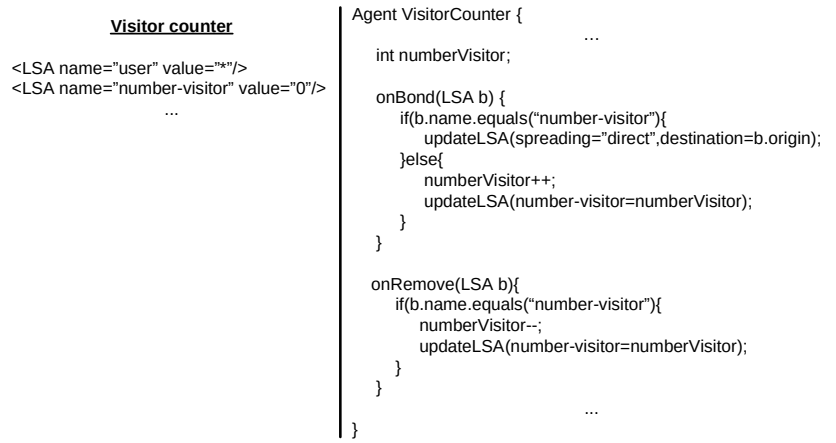

Fig. 9. Injected LSAs (left side) of the visitor counter agent and partial code of *onBond()* and *onRemove()* methods (right side) of the visitor counter agent.

the solution proposed with MalacaTiny-Sol because it has not to rely on third components to provide the position of visitors in the IM. When a visitor enters to a new room, his/her agent injects an LSA in the SAPERE node with the personal preferences of the user. On the other hand, agents associated to exhibit have injected LSAs with information about the exhibit. When eco-laws are fired, the user's LSA is bonded to the exhibit LSAs of interest for him/her and the information is presented to the visitor.

3.2 Scenarios of Emergency Evacuation Planning

In this section, we are going to model a service that contributes to the evacuation of the building in case of emergency in both approaches. In order to illustrate the advantages and disadvantages of both, we are going to consider in this scenario two situations: if there is just one emergency exit; and when there are more than one emergency exits available.

Designing applications with MalacaTiny-Sol

One emergency exit application. The evacuation starts when a member of the security staff detects an emergency situation. Firstly, when an emergency is detected, the security guard agent of the person that detects it, notifies all the people in the IM that there is an emergency situation. To make this notification more efficient, group-based communication is used again. In this case a message is sent to the group of visitor agents and another to those composed by security agents. With the information provided in the message, visitor agents plan how to get to the emergency exit while avoiding the site of the emergency. Security guards agents use this message to inform the security staff of where the emergency exists and what kind of emergency it is.

In order to implement this behavior in the security guard agent, new aspects are added (see Fig. 4): the joint work of the *LocationProvider* component and the *LocationUpdater* aspect estimates and updates the user position that is going to be used in the emergency message; and finally, the *EmergencyProtocol* aspect continues with joining the agent to the group of security guard agents, sending an emergency message to the two groups of agents and also receives emergency messages. The design of the visitor agent also requires more elements (see Fig. 5) to manage an emergency situation: the *EmergencyProtocol* aspect receives emergency notifications from security guards and updates the internal knowledge of the agent activating an emergency situation; when this occurs, the *RouteMonitor* aspect requests a route from the *RoutePlanner* component and when the route to the emergency exit is generated, it guides the user to the exit using his/her current location.

Multiple emergency exits application. When there is more than one emergency exit to choose from the situation is similarly handled. As in the previous case, the corresponding security agent sends the message to the group of security agents with the same result. Additionally, it has to determine the number of visitors and their position in the IM. In order to get this information, a message requesting the position of the visitor agents is sent using the group-based communication. When the security agent receives the information it assigns an emergency exit to the visitor according to their current position and the number of visitors in the same room (in order to ensure speedier the evacuation). To implement this behavior the design of the agent for visitors is not modified, but the security guard agent needs to change the behavior of the *EmergencyProtocol* and add a new component for planning the visitors evacuation (*EvacuationPlanning*).

Designing applications with SAPERE

One emergency exit application. The emergency planning in SAPERE uses the work of the spread and aggregate eco-laws to enable a field-based coordination mechanism [17] that notifies of the emergency and indicates the exit path simultaneously. To trigger this process the agent associated with the guard has to inject an LSA in the SAPERE node located at the emergency exit. This LSA is spread to the other nodes hop-by-hop starting with those that are directly connected to the space in which the LSA was initially injected. When this LSA is spread to other SAPERE nodes, the attribute field is set to the previous node and the hop counter is increased. If multiple emergency LSAs are spread to the same SAPERE node with different origins (i.e. different values of previous attribute), when the aggregate eco-law is fired, only the LSA with the minimum hop-counter remains (see attribute aggregation in Fig. 10). In order for the visitor to receive the emergency notification, his/her associated agent has to inject an LSA in order to receive emergency notifications. The bonding of this LSA enables the planning of the emergency route step by step. Each time the LSA is bonded, the visitor receives a notification of the next room that he/she has

to reach to get to the exit of the building. When the user reaches the specified room, the same process is repeated. In this way, visitors find the emergency exit. The application of the aggregation eco-law ensures that visitors always follow the path with the minimum number of hops to the exit.

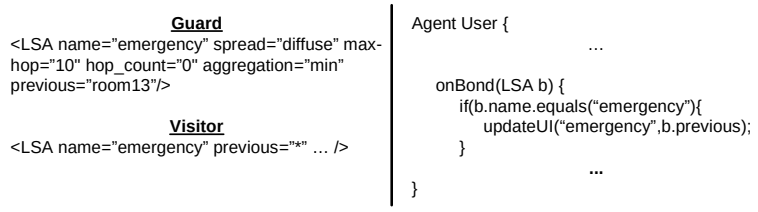

Fig. 10. Injected LSAs (left) of agents for visitors and guards and partial code *onBond()* method (right) of the agent for visitors in the emergency evacuation.

Multiple emergency exits application. In the case of multiple emergency exits, security guard agents inject an LSA with the same format as in the previous case and the path with the minimum number of hops is also ensured by aggregation eco-law. The problem is that with this schema we cannot control the number of people that are sent to the different exits. This is because we cannot ensure, on the one hand the minimum path (applying the aggregation eco-law) and on the other hand, to have multiple options that can be used for a specific agent to send a person to one exit or another.

4 Comparison

In this section we are going to evaluate and compare the systems resulting from the design of the scenarios described in the previous section and modeled using MalacaTiny-Sol and SAPERE. The reason for this comparison is to measure the advantages and the benefits of both approaches from a software engineering point of view. Specifically, this assessment focuses on the design of the security guards and visitor agents. For the evaluation, we are going to use a combination of the frameworks provided by [18, 19]. On the one hand, the work in [18] provides an architectural metric suite that is being widely used to measure the separation of concerns in software systems. On the other hand, the paper [19] presents a framework for the evaluation of ubiquitous computing systems, which can be used to evaluate pervasive applications as is our case study. Specifically, we are going to evaluate: (i) Separation of Concerns (SoC) - 4.1; (ii) Coupling and Cohesion - 4.2; (iii) Adaptivity - 4.3; (iv) Robustness - 4.4; (v) Scalability - 4.5; and (vi) Privacy - 4.6.

In order to implement the metrics for measuring SoC, coupling an cohesion, we are going to use common concerns of MalacaTiny and SAPERE. They are

representation, distribution, coordination, context-awareness, bonding, computation and spreading. The mapping between them is depicted in Fig. 11.

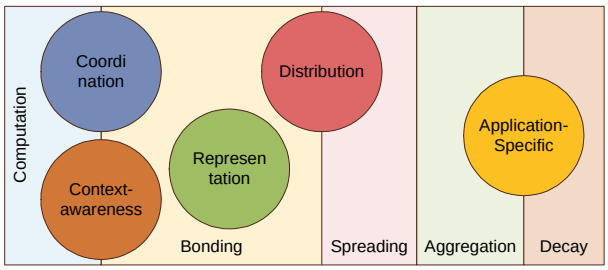

Fig. 11. Mapping between MalacaTiny (circles) and SAPERE concerns (rectangles).

4.1 Separation of Concerns

SoC is a well-established principle in software engineering which aims to improve the internal modularity and maintainability of the crosscutting concerns of a software design. A crosscutting concern is a special concern which naturally cuts across the modularity of other concerns. Without proper means for separation and modularization, crosscutting concerns tend to be scattered and tangled up with other concerns. The natural consequences are reduced comprehensibility, ease of evolution and reusability of software artifacts, which limit the adaptability, robustness and scalability of the software system.

A way to measure the degree of SoC is to quantify the diffusion of a concern over components, interfaces and operations. Concern Diffusion over Architectural Components (CDAC), Interfaces (CDAI) and Operations (CDAO) measure the degree of concern scattering at different levels of granularity. The results of the SoC metrics are obtained for each concern of interest in the system. The metrics for computing the separation of architectural concerns are applied to calculate the degree to which a single concern or property of the system maps to the architectural components.

The results of the assessment show SAPERE scores better for CDAC and CDAI, while MalacaTiny is better for CDAO (see Table 2). In SAPERE, the agent class encapsulates all concerns but bonding and spreading, that are in the SAPERE node, so their values for CDAC are 0 or 1. In MalacaTiny each concern is encapsulated as an independent aspect, and there is a coordination aspect for each interaction in which the agent participates and context aware behavior (see Figures 4 and 5). The results for the bonding concern for MalacaTiny are because this concern includes coordination, context-awareness, distribution and representation (see Fig. 11). Additionally, the agent interaction is supported in MalacaTiny by 1 interfaces and in SAPERE by 1.

Table 2. SoC measurements for MalacaTiny-Sol (M-S) and SAPERE.

Concern	CDAC		CDAI		CDAO	
	M-S	SAPERE	M-S	SAPERE	M-S	SAPERE
Context-awareness	2.5	1	2	1	2	9.5
Coordination	2.5	1	2	1	3	9.5
Distribution	1	1	2	1	2	4
Representation	1	1	1	1	2	4
Bonding	7	0	2	0	9	0
Computation	5	1	2	1	5	9.5
Spreading	1	0	2	0	2	0
Average	**2.85**	**0.7**	**1.85**	**0.7**	**3.57**	**5.2**

Finally, the interception point model of MalacaTiny and the number of methods that use SAPERE to receive results of the eco-laws application explain CDAO results. MalacaTiny has 3 interception points and each aspect has at least 1 method to access the aspect behavior. For example, the coordination aspect requires 3 operations because it is affected by 2 interception points (i.e. reception and sending of the messages), and has 1 method to access its behavior. On the other hand, SAPERE agents have 4 four methods to receives results from the SAPERE node, 4 operations to interact with it and additionally, we have to consider methods in the agent class that call to these operations (see Fig. 3). For example, for the computing concern, the agent for guards scores 10 and the agent for visitors 8.

The results of these metrics describes the main features of the architectures of agents in FIPA-based and tuple-based approaches. Agents in FIPA have a more complex design because the negotiation (coordination in tuple-based approaches) is accomplished inside the agent, so CDAC and CDAI is always higher in these approaches. The results for CDAO are a consequence of the complex interaction that tuple-based agents have with their middlewares. Therefore, despite MalacaTiny-Sol scores are good, considering that it is a FIPA-based approach, SAPERE gets a better SoC.

4.2 Coupling and Cohesion

Coupling and cohesion are two quality attributes of a software design that reflect the quality of a good modularization. Coupling refers to the level of interdependency among the modules (e.g. components) and cohesion is the level of uniformity of concerns of a single module (i.e. the degree of relatedness among the elements -attributes, methods- of a component). A high degree of coupling drastically reduces component reuse, which in turn means poor adaptability. Low cohesion means a concern is spread over different modules, and its evolution as

an independent entity will therefore be very difficult to manage. Consequently, it is important to minimize coupling and maximize cohesion in the system design.

The coupling metrics measure the number of components connected to each other. Coupling is evaluated using the Fan in and Fan out metrics for each element of the SAPERE and MalacaTiny agents. These metrics count the number of conventional components which require services from the assessed component (Fan in metric) and the number of components from which the assessed component requires services (Fan out metric).

Table 3 shows the average for coupling and cohesion measurement per component in both architectures (rows labeled "Average") and the percentage difference of each metric between MalacaTiny-Sol and SAPERE (row labeled as "Percentage Difference"). The positive values of the "Percentage Difference" means lower results for SAPERE, while negative results means lower results for MalacaTiny-Sol. In this case, SAPERE scores better for Fan in because SAPERE agents directly perform less functionality than MalacaTiny agents. On the other hand, MalacaTiny scores better for Fan out because the application specific components of agents have a lower value for this metric.

Table 3. Coupling and Cohesion measurements for MalacaTiny-Sol and SAPERE.

MalacaTiny-Sol	Fan in	Fan out	LCC
Application specific components	1	0.55	0
Context-awareness aspect	1	1	3
Coordination aspect	1	1	3
Distribution aspect	1	1	2
Representation aspect	1	1	2
Core agent	9	1	0
Average	2.33	0.925	1.66
SAPERE	**Fan in**	**Fan out**	**LCC**
Application specific components	1	1	0
Core agent	1	1	5
Average	1	1	2.5
Percentage difference	80%	-7.79%	-40%

Cohesion is measured using the Lack of Concern-based Cohesion (LCC). This metric counts the number of different system properties addressed by each class (in SAPERE agents), components and aspects being considered (in MalacaTiny agents). For this metric MalacaTiny scores better (see Table 3). This is because the Malaca architecture focuses on the separation of concerns at the agent level, while SAPERE applies the separation at infrastructure level. So, in the SAPERE agent class all the concerns used in the evaluation are contained with the exception of bonding and spreading that are in the SAPERE node.

With these results we can again see reflected, the architectural features of both type of approaches. MalacaTiny successfully exploits its AO to offer agents that scores better in Fan out and LCC. Scores of MalacaTiny mean a high reusability of the internal components and aspects of agents of this scenario. The good results of SAPERE are supported by the lower number of components required to develop SAPERE agents. Therefore, on the one hand MalacaTiny-Sol efficiently handles with complex designs, on the other hand, with SAPERE such type of designs are not necessary.

4.3 Adaptability

The adaptability metric measures how the system adapts to changes that are external to the application, i.e. changes in user preferences, devices and in the physical space where the IM is located.

The recommendations of exhibits of interest for users depending on their location in the IM (see Subsection 3.1) can be useful for new users and annoying for users that know the exhibition. So, a useful functionality that the applications can provide is disabling this service when users request it. To do so, MalacaTiny requires to modify aspect composition rules (see Subsection 2.1) for not applying *ExhibitRecommender* aspect (see Fig. 5). SAPERE accomplish the same task by not injecting LSAs with user preferences. These procedures can be applied to disable any service that agents provide or consume in both systems.

Sol offers to the agents deployed on it the possibility of changing the network interface used for their connection to the AP. As stated before, Sol has support for multiple network interfaces (see Subsection 2.1). An agent can connect to Sol using a WiFi connection, but the agent can change the network interface to Bluetooth in case of bad coverage. Accomplishing this task does not just require a change in the composition rules. Additionally, the agent has to interact with the AP in order to ensure that it remains in the same groups and provides the same services.

Changes in the museum map have different consequences for the solutions proposed in both approaches. In MalacaTiny, some of the agent services depend on the component *LocationProvider* (see Figures 4 and 5). This component informs the room where user is currently and depending on its implementation, it is necessary its substitution. SAPERE agents rely on the location to provide services too. The extension of the museum map requires the addition of new SAPERE nodes to these new locations and a change of the routing tables of some SAPERE nodes. The design of the agents remains the same, but the agent for guards has to update its internal knowledge about the IM (see Subsection 3.1).

The modification of the map impacts the emergency evacuation (subsection 3.2) in MalacaTiny. In order to ensure a correct planning of the exiting route, it is necessary to modify the *RoutePlanner* component inside the agent for visitors. If the IM has a new emergency exit, it is also necessary to change the *EvacuationPlanning* component. On the other hand, in SAPERE, with the modified

infrastructure, the implementation of the emergency planning inside agents remains the same.

Table 4 summarizes the main changes in each of the agent systems to adapt agents and infrastructures. In conclusion, both approaches (FIPA-based and tuple-based) can easily adapt the set of services provided by their agents. MalacaTiny offers more possibilities to adapt the agent architecture. Finally, the adaptation of location-aware services to changes in the physical space requires an extra effort in MalacaTiny, in the case of SAPERE this effort is made at the infrastructure level and requires only little modifications in agents.

Table 4. Issues to change when adaptation is required in MalacaTiny-Sol and SAPERE.

Adapted issue	Services provided	Network interface	Deployment space
MalacaTiny-Sol	Composition rules	Composition rules / interaction with Sol	Components and aspects that depend on location
SAPERE	Injected LSAs	Not possible	SAPERE nodes

4.4 Robustness

While adaptability measures how the application deals with external changes, the robustness metric measures how internal events affect the application or the percentage of faults that are invisible to user. In both approaches there can be faults both at application level and at infrastructure level. In the case of the application level, if a security guard agent or a visitor agent stops, both MalacaTiny and SAPERE users notice that something is going wrong. However, if the agent that fails is an exhibit one, it only results in the information of the associated exhibit not being presented to the user but the application keeps running. This can also be applied to sensors (see subsection 3.1).

The robustness of the visitor counter scenario (see subsection 3.1) is similar in both approaches. In the case of MalacaTiny-Sol, the group mechanism supports the provision of this function. Group communication principally uses IP multicast, which is based on UDP at the application level and is an unreliable transport protocol. This means that message reception is not ensured. This issue can affect the accuracy of the number of visitors obtained in MalacaTiny-Sol. In the case of SAPERE, counting the visitors depends on the visitor counter agents deployed in SAPERE spaces. The accuracy of the information provided by these agents depends on the coverage of the SAPERE spaces and the position of the visitors in the IM. So, as in the case of MalacaTiny, the security guard only receives an estimation of the number of visitors. Additionally, in SAPERE if the visitor counter agent fails, the security guard agent will not receive the

information from the room where this agent was deployed. Therefore, the distribution of the functionality between agents and AP in MalacaTiny-Sol provides a more robust application.

At infrastructure level, SAPERE is more robust than Sol. The Sol AP usually runs in a single node with multiple clients that depend on it (see Fig. 1), if the AP fails then those agents cannot interact, register services and join groups and the IM cannot offer services to any of its users. The restarting of the Sol AP affects the entire MAS. However, the distribution of SAPERE nodes causes that the failure of a node only affects users that are in the same room (see Fig. 2). Additionally, if the node restarts, only the agents associated with this physical space (agents for sensors, exhibits, to gather environmental conditions and for counting visitors) are affected.

In conclusion, in the IM scenario the distribution of the functionality between agents makes the solution based on FIPA more robust. On the other hand, at infrastructure level both middlewares are special cases. In the case of Sol, their current implementation does not offer a robust infrastructure, but other FIPA-approaches, like Jade, handle the eventual failure of the main node of the middleware. SAPERE is a special case in tuple spaces, given that they usually present a centralized distribution. Therefore, and in order to enhance the robustness of these approaches, the infrastructure of Sol should be modified to make it as robust as other FIPA approaches, in the manner that SAPERE offers a more robust infrastructure, unlike similar coordination approaches.

4.5 Scalability

The scalability metric measures how the complexity of the system increases when some feature is extended or the system must met a new requirement. In agent approaches, the extension of a system means the addition of new agents or the modification of an existing one. In this section, we are going to study the scalability of these two agent systems, then studying the effort required to extend the system.

The effort required to add a new agent in the MAS is related with the number of elements that compose the agent and the component reuse. SAPERE agents requires less elements (a mean of 12.5 in MalacaTiny vs. 2 in SAPERE) but the component reuse is higher in MalacaTiny (3 in MalacaTiny vs. 0 in SAPERE).

The extension of an agent to meet new requirements is usually related with the addition of new services to agents other than those which we have initially considered in our design. In this subsection we are going to consider 3 kind of services: (i) with a single provider, (ii) with multiple providers; and (iii) global services. Service discovery, invocation and provision is easier in SAPERE because of the application of the bonding eco-law. What in MalacaTiny-Sol is done in 3 steps (query the DF of the Sol AP, request the service and consume the service) in SAPERE is done in 2 steps (inject the LSA and receive the service).

In MalacaTiny-Sol, the addition of a service with a single provider requires the addition of new aspects and components, and the modification of the aspect composition rules. In SAPERE, this requires the injection of new LSAs and the

modification of the agent class to provide or consume the service. To promote the code reuse in SAPERE, the class of the initial agent is extended. In both approaches this extension just implies the modification of the core agent class, however these modifications are easier in MalacaTiny because it ensures the code reuse. To the contrary, the code reuse is more problematic in SAPERE, and consequently so is the extension of the agent. This is because we cannot extend the agent with services provided by more than one agent because multiple inheritance is forbidden in Java. Moreover, to use directly the code is difficult because this is spread between the different methods of the agent (see Table 2).

The addition of a service with multiple providers is different in both approaches. In MalacaTiny-Sol, agents need a protocol to select the most adequate provider according to some criteria (e.g. using a FIPA Contract Net) which makes the design of the coordination aspect more complex. On the other hand, the case of SAPERE is simpler because in some cases is the LSAs space which selects the most adequate service provider. If the selection criteria is numerical and the most adequate service provider is that which has the minimum or the maximum value, then the LSA space selects the most adequate service provider means of the aggregation eco-law. If the criteria is not numerical and only implies the existence of a specific feature, it is necessary to include this feature in the LSAs that publish and request the service. If more than one LSA shares this feature, one of them is randomly selected for bonding.

In the case of global services, i.e. services available in any room of the IM, the design of the MalacaTiny does not require any special consideration, but the the SAPERE solution must be modified. This was illustrated in the case of the visitor counter application, that has to deploy purpose specific agents in each SAPERE node. These agents gather the information related with the service and send it (under request) to the interested agents.

The explanations in this section exemplify the work of both coordination approaches in scalability. Without a proper modularization that promotes the code reuse, the addition of new agents requires less effort in tuple-based approaches because the design of agents is simpler. On the other hand, the extension of agents, which usually implies the addition of new services provided or consumed by them, is usually easier to design in FIPA approaches, while in tuple-based approaches the provision and consumption is easier. In the case of these two agent systems, we can conclude that the solution provided by MalacaTiny-Sol is more scalable. MalacaTiny offers a uniform solution for the extension of agent capabilities and promotes software reuse which decreases the development effort. The strongest point in favor of SAPERE is the development of services based on local interactions which are provided and consumed means of the bonding eco-law, which are very likely to be found in pervasive environments.

4.6 Privacy

The privacy metric evaluates the type of information that the user has to provide (and divulge) in order to profit from application, and the availability of the user's information, for other users of the system as third parties. In the scenarios

presented, users share three types of data information: presence, location and personal profile. Table 5 depicts what kind of user information is shared by the security guard agents (G) and the visitor agents (V) in the different scenarios. According to the table, scenarios modeled in MalacaTiny-Sol require less information to be shared (user profile remains inside the personal user agent) than those modeled in SAPERE. Additionally, in the case of MalacaTiny the personal information of the users is located in their personal devices, while in the case of SAPERE this information is located in these devices and in SAPERE spaces, where the information can be accessed (through bonding) by all the agents of the IM.

Therefore, we can conclude that MalacaTiny-Sol scores better in privacy because users divulge less information to obtain value from the application and the availability of the information to other users is lower. These scenarios illustrates the work of FIPA-based and tuple-based approaches in privacy. Local computation of FIPA approaches makes easier to ensure the privacy of users.

Table 5. Type of information shared by agents for guards (G) and for visitors (V) in scenarios of the IM.

Scenario	MalacaTiny-Sol			SAPERE		
	Presence	Location	Personal	Presence	Location	Personal
Environmental monitoring						
Visitor counter	V			V	G	
Information provision						V
Single emergency exit		G		G,V		
Multiple emergency exit		G,V		G,V		

5 Conclusion

In this paper we have presented the modeling of a classical pervasive scenario, an IM, using two agent systems for pervasive computing, MalacaTiny-Sol and SAPERE. The first one is based on FIPA interaction protocols and the second one is based on tuple spaces. The resulting systems have been evaluated using an architectural metric suite that measures SoC, Coupling and Cohesion, and additionally, we have discussed other important concerns in pervasive systems such as adaptability, robustness, scalability and privacy.

Results of the architectural suite are specific to this case study, but these highlight the architectural features of both agent technologies and also support the following argument. The benefits from both approaches for the development of pervasive application are not only from their schemas of interaction, also from their mechanisms for ensure SoC (i.e. AO vs. eco-laws) and how they adapt the agent paradigm to pervasive computing (e.g. groups vs. distributed tuple spaces).

In MalacaTiny-Sol, the design of the agents is more adaptable, scalable and can ensure the privacy of users easily. FIPA-based approaches allow the set of offered services to be modified by enabling or disabling behaviors that perform such services. MalacaTiny-Sol does this efficiently even at runtime because these behaviors are encapsulated in aspects whose functionality is driven by a set of composition rules. Additionally, MalacaTiny agents can utilize the multiple network interfaces offered by Sol because of the AO. This is an advantage for the development of leisure applications where users with different types of network technologies in their devices (e.g. Bluetooth or WiFi) can interact with the same system. In general, FIPA-based approaches offer solutions that are more scalable because the extension of the system is uniform. Additionally, because the computing is encapsulated inside agents the risk of lateral effect when the system is extended is lower than in tuple based approaches such as SAPERE. Finally, in FIPA-based approaches it is easier to ensure the privacy of users, because the most of the computation is performed locally.

SAPERE agents have a good capacity for adaptation too, their service consumption and provision is easier and additionally, the resulting system is more robust. While in FIPA-approaches adaption of services is related to the number of protocols that agents can handle, in tuple based approaches the relation with the number of tuples which are injected in the space. So, the adaptation of both types of approaches presents a similar difficulty. The distributed nature of SAPERE applications results in systems that adapt easily to changes in the physical space where the application is distributed. In tuple based approaches the service provision and consumption is more efficient than in FIPA ones because a direct interaction between agents is unnecessary. The negotiation is done in the tuple space via a pattern matching process that avoids message exchange. Finally, SAPERE spaces offers a more robust infrastructure thank to its multiple SAPERE nodes. This is not a common feature in tuple based approaches but it is one of the strongest points in SAPERE.

Both approaches can be combined with benefits to both. The AO of MalacaTiny agents and its extensible join point model makes the deployment of these agents in SAPERE nodes possible. In this combination, MalacaTiny becomes a tuple-based agent. To do this, it is necessary to develop a distribution aspect for SAPERE and to add 4 interception points in the agent that correspond with bonding, reading, removing and updating of the LSAs. The main benefits for MalacaTiny is the usage of an infrastructure which is more robust than Sol and offers a natural metaphor to develop services that depend on the location of users. The main benefits for SAPERE would be to enhance the internal modularization of agents deployed in SAPERE nodes, that promote reuse and ease the adaptation of agents even at runtime. It is interesting to note that these benefits are not related with features of the coordination approaches, but rather with adaptations of these agent technologies to the pervasive computing environment. As future work, we plan to study the combination of these approaches.

Acknowledgment

Work supported by the Andalusian regional project FamWare P09-TIC-5231, the European projects INTER-TRUST FP7-317731 and SAPERE FP7-256873, and the Spanish Ministry Projects RAP TIN2008-01942 and MAVI TIN2012-34840.

References

1. Saha, D., Mukherjee, A.: Pervasive computing: a paradigm for the 21st century. Computer **36**(3) (mar 2003) 25 – 31
2. Henricksen, K., Indulska, J., Rakotonirainy, A.: Modeling context information in pervasive computing systems. In: Pervasive Computing. Volume 2414 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2002) 167–180
3. Sadri, F.: Ambient intelligence: A survey. ACM Comput. Surv. **43**(4) (October 2011) 36:1–36:66
4. Cook, D.J., Augusto, J.C., Jakkula, V.R.: Ambient intelligence: Technologies, applications, and opportunities. Pervasive and Mobile Computing **5**(4) (2009) 277 – 298
5. Omicini, A., Denti, E.: From tuple spaces to tuple centres. Science of Computer Programming **41**(3) (2001) 277 – 294
6. FIPA: The Foundation for Intelligent Physical Agents. `http://www.fipa.org/`
7. Ayala, I., Amor, M., Fuentes, L.: An agent platform for self-configuring agents in the internet of things. In: Third International Workshop on Infrastructures and Tools for Multiagent Systems. ITMAS 2012. (June 2012) 65–78
8. Castelli, G., Mamei, M., Rosi, A., Zambonelli, F.: Pervasive middleware goes social: The sapere approach. In: Fifth IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW). (oct. 2011) 9 –14
9. Labrou, Y., Finin, T., Peng, Y.: The current landscape of agent communication languages. Intelligent Systems **14** (1999) 45–52
10. Ayala, I., Amor, M., Fuentes, L.: Self-configuring agents for ambient assisted living applications. Personal and Ubiquitous Computing (2012) 1–11
11. Amor, M., Fuentes, L.: Malaca: A component and aspect-oriented agent architecture. Information and Software Technology **51**(6) (2009) 1052 – 1065
12. Oracle: Sun SPOT world. `http://www.sunspotworld.com/`
13. Libellium: Waspmote. `http://www.libelium.com/products/waspmote`
14. Libellium: Meshlium Xtreme. `http://www.libelium.com/products/meshlium`
15. Mamei, M., Zambonelli, F.: Programming pervasive and mobile computing applications: the tota approach. ACM Trans. Software Engineering and Methodology **18**(4) (2009)
16. Rodriguez, M., Favela, J., Martinez, E., Munoz, M.: Location-aware access to hospital information and services. IEEE Transactions on Information Technology in Biomedicine **8**(4) (dec. 2004) 448 –455
17. Mamei, M., Zambonelli, F.: Field-Based Coordination for Pervasive Multiagent Systems. 1st edn. Springer Publishing Company, Incorporated (2010)
18. Sant'Anna, C., Lobato, C., Kulesza, U., Garcia, A., Chavez, C., Lucena, C.: On the quantitative assessment of modular multi-agent system architectures. NetObjectDays (MASSA) **224** (2006)
19. Scholtz, J., Consolvo, S.: Toward a framework for evaluating ubiquitous computing applications. IEEE Pervasive Computing **3**(2) (2004) 82–88