1

Intelligent Agents for Automatic Service Composition in Ambient Intelligence

Maria J. Santofimia, Francisco Moya, Felix J. Villanueva,
David Villa and Juan C. Lopez
Universidad de Castilla-La Mancha
Spain

1. Introduction

The Ubiquitous Computing concept was first defined by Mark Weiser in (Weiser, 1995), referring to a new computing era where electronic devices merge with the background, becoming invisible, in such a way that people could make use of those devices in an unconsciously way, focusing just on their needs and not in the interaction.
One decade later, the IST Advisory Group first states the concept of Ambient Intelligence (Ducatel et al., 2001), which lying on the ubiquitous computing paradigm, refers to those environments where people are surrounded by all kind of intelligent intuitive devices, capable of recognising and responding to their changing needs. In these contexts, people perceive the surrounding as a service provider that satisfies their needs or inquiries in a seamless, unobtrusive, and invisible way.
These computing paradigms set a frame of reference, characterised by being mainly concentrated on releasing mechanisms that gather information about users, match behavioural patterns, or predict user actions, requirements and needs (Costa et al., 2007) (Cugola & Picco, 2006) (Issarny et al., 2005) (Prete & Capra, 2008). Nevertheless, the Ambient Intelligence paradigm is meant to consider users as constituent parts of the context, although in most solutions presented to date, users are considered in isolation. In this regard, extending the user-centered view, in order to encompass the context services and purposes, arises as key requirements for systems in Ambient Intelligence.
It soon becomes apparent the need for a multidisciplinary approach capable of addressing all the emerging challenges. One of these fields is concerned with the communication support. The heterogeneity of the context devices, as well as their dinamism, impose high demands upon the middleware platform, that it is now responsible for abstracting the technological peculiarities. It is then possible to provide a common and well-known set of communication interfaces. These interfaces are described in terms of a semantic model, that can be easily shared and translated into different languages, so that in can be used by the rest of involved technologies (intelligent agents and reasoning engine). Finally, and probably the most important part of the provided solution, refers to the context-awareness, in charge of understanding the context. This requires some approach that resembles human behaviour in regard to its ability to deal with information of an imprecise nature, ambiguous, and of a questionable retrievability, but also capable of making decisions based on this partial information. To this

end, involving intelligent agents, seems to be an effective approach when resembling human behaviours.
Accordingly, this work presents a comprehensive approach towards Ambient Intelligence, making special emphasis on the role played by the intelligent agents, as the key element orchestrating the overall solution.

2. Background Information

This section basically describes the background information behind the combined proposal presented here as an approach towards Ambient Intelligence.

2.1 Service Oriented Architecture

The service concept, regarding the Service-Oriented paradigm, is coupled to whatever the resource (distributed object, agents, or basic web services, for example), in such a way that one service is created for each available resource. This paradigm identifies two types of services: the basic and composite ones. The former are the services directly offered by devices, while the later are composed of basic service aggregations, which result in more complex services.
Service-Oriented Architectures (SOAs) support the development of highly adaptable systems, capable of adding, removing and replacing the constituent services of the system. Commercial standards developed with this purpose in mind already exist, such as Web Services. However, despite the achieved success , composition is currently a subject of study for ongoing research and is still far from a mature stage.
Most of the Service Oriented Architecture (SOA) systems for Ambient Intelligence simply pursue the system reconfigurability or adaptability, placing the responsibility to do this on the middleware framework. Obviously, these systems cannot take decisions out of the prefixed behaviours, set beforehand, neither can they tackle unforeseen circumstances, requirements and needs. Therefore, instead of trying to describe the responses to all the likely scenarios, it is more effective to enumerate the main goals that the system has to achieve or maintain, along with the basic mechanisms available to this end. The service composition paradigm provides the foundations for generating new services that fulfill new needs or requirements. However, in the context of Ambient Intelligence, this task should be automatically accomplished, relieving the user from dictating the basic services involved in the composition.

2.2 Approaches towards Automatic Service Composition

The most relevant approaches intended to provided systems for Ambient Intelligence with the capability to automatically compose services are described underneath:

- Configuration files: The first attempt to provide some sort of dynamism was based on the adaption concept. This proposal was intended to provide architectures with adaption capabilities by means of configuration files. These architectures counted on some features that could be customised in order to select the components loaded at startup time. This approach allows a limited level of dynamism, far from what an adaptive middleware should be, since the main features are placed at the middleware kernel and could not be changed. Therefore, rather than adaptive middlewares they should be considered customisable middlewares, supporting just a fixed group of cases.
- Reflection: As a second attempt, after the use of configuration files, reflectiveness appeared as the solution to add some dynamism support to middlewares. This proposal advocates for a middleware core with a minimal set of services installed in devices.

By means of reflective mechanims, applications can obtain from the middleware the context information, and use it to tune the middleware behaviour.

- Reflection and metadata: The next stage in this evolution is based on the combination of reflection and metadata, aimed at developing adaptive and context-aware applications. This approach is mainly based on policies, that is, the use of a set of primitives aimed at describing how the context might change and how these changes are to be treated. Since conflicts among policies may arise, a solution based on a micro-economic approach was proposed in order to handle this conflicts.
- Externalization: Although reflective middleware services do support configurability, by supporting replacement and assembly of components in reaction to changes, the reality was that most of them assumed a basic backbone of fixed services. The externalization approach advocates for a middleware architecture that explicitly externalises the state, the logic, and the internal structure of middleware services, in such a way that the system can be updated, upgraded, or changes its configuration without requiring user intervention.
- Policies: This approach proposes the use of profiles, where the associations between services and policies applied to these services are described. Profiles are passed down to the middleware, and whenever a service is invoked, the middleware consults the profiles of the application that requests it. The profile determines which policy can be applied in the current context, depending on the state of the requested resource, thus relieving the application from performing these steps.
- Web Services: Among all these different approaches towards service composition, Web Services have been by far the most popular. This XML-based approach allows the specification of web services that can be dynamically loaded according to the requests. A service is specified by means of a service abstract interface and the non-functional properties associated with the service. This approach provides a set of Service Repositories containing information about local and remote service repositories.
- Ontologies: Finally, it has to be pointed out that nowadays, the use of ontologies is gaining great attention. Among all the ongoing proposals on these field, domain ontologies is one of the most relevant, and is intended to model the domain knowledge and provide semantics to service description. The capability to express semantic relations among services is quite useful in guiding the composition process.

2.3 Intelligent Agents

The motivation behind the use of an agent-based approach is founded on the possibility to describe agents as goal-oriented entities, on the basis of the BDI model presented in (Bratman, 1987), and developed in (Rao & Georgeff, 1991) as an approach to model rational agents, using three basic mental attitudes, such as beliefs, desires and intentions.

Nonetheless, different approaches to implement the intelligent agents can be found in literature (Wooldridge, 2000): Logic based architectures (deductive agents), reactive architecture (reactive agents), layered architectures (hybrid agents), and practical reasoning architectures (the aforementioned Belief-Desire-Intention agents). Among these alternatives, the Belief-Desire-Intention model (BDI) has proved to be a powerful framework for building rational agents.

The BDI model of decision making is intended to reproduce the process carried out when people make decisions in order to achieve a certain goal. The main characteristic of the BDI

model lies on the significance conceded to beliefs, desires, and intentions involved in rational actions. Therefore, those systems that grant importance to these attitudes over any other, are often referred to as BDI-architectures. Beliefs are the information agents hold about the world, which is not necessarily accurate. This information might change as a result of new perceptions or the execution of intentions. Desires or goals refer to those tasks that, in an ideal world, the agent would like to accomplish. Intentions are those desires that agents are committed to accomplish.
The Jadex framework (Pokahr et al., 2005) provides an agent-oriented reasoning engine that also supports the development of rational agents. In spite of using formal logic descriptions, Jadex proposes the use of two commonly known languages, such as Java and XML. The BDI agent is modeled by mapping the concepts of beliefs into Java objects, while desires and intentions are mapped into procedural recipes coded in Java that the agent carries out in order to achieve a goal.

3. The Architecture Foundations

Having an accurate semantic model seems essential for an architecture intended to support Ambient Intelligence contexts. This semantic model is shared with the agent-based platform and the reasoning system, supporting cooperation among them. Figure 1 depicts the overview of the proposed solution to support Ambient Intelligence contexts. The different technologies proposed are intended to tackled the challenges arising in such contexts. Therefore, the middleware module is in charge of managing the services deployed in the context, basically by supporting the communication with the services. Furthermore, the intelligent agents interact with the environment, not only by gathering information but also by accomplishing actions by means of actuators. Furthermore, in order to be context-aware, the role played by the reason-

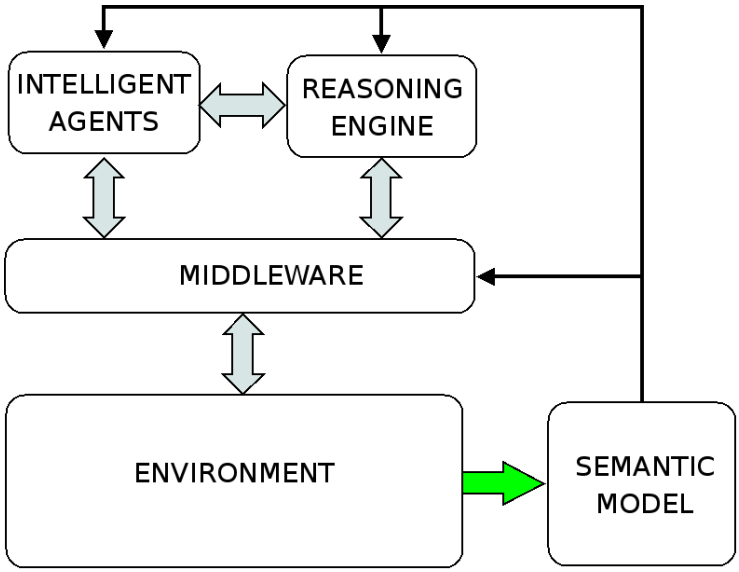

Fig. 1. A comprehensive view of the propose architecture.

ing engine fulfills this purpose by reasoning about the information gathered by the intelligent agents.
However, these three modules need to be connected in order to cooperate towards supporting the Ambient Intelligence contexts. To this end, the semantic model supposes this linking element, that is common to all the modules of the architecture and provides a way of communicate them.

3.1 The semantic model for Ambient Intelligence

Ambient Intelligent contexts are characterised for being highly dynamic and heterogeneous in terms of provided services, existing users and devices. In such scenarios it is hard to accomplish a thorough description of the context information and pretend it to be up to date along the time. For such reason, it is preferable to provide the architecture with the capability to describe the context on the basis of a semantic model injecting meaning to the context information.
Once the importance of an accurate semantic model has been stated, next step consists of selecting the right approach to undertake the model description, and among the many different approaches (Strang, 2004), ontologies are a widely accepted technique to accomplish semantic descriptions, in terms of the relevant entities of the domain and the relationships established among them. There exist several formalisms for representing ontologies, however, the Web Ontology Language (OWL), (encoded as a RDF/XML) is widely used and supported with tools such as Protégé (BMIR, 2009) that simplifies the tedious task of describing an ontological model.
Nevertheless, the graphical models that can be derived from the ontology, using Protégé or any other tool, are commonly poor, and relationship others than "is-a" are hard to catch at a glance. Considering that ontologies simply state entities and relationships among entities, resort to the Entity-Relationship (E-R) diagram seems to provide a more understandable graphical representation, as it can be seen in figure 2.
The semantic model here presented has been intentionally designed to support automatic service composition, and for this reason, services are semantically described by means of specifications, which at the same time are made of properties. These specifications, as it will be explained later, are intended to support the reasoning process that leads the service composition. Furthermore, this semantic model also describes services in terms of the actions the service performs, and the object over which such actions are accomplished. Each action is performed on an object, therefore, services are described as a set of actions performed over certain objects. For instance, an authentication service, performs the action of authenticate over the object of domain user type.
The benefits of using such a semantic model, rather than a simple service taxonomy, as the one provided by UpnP, for instance, lays in the fact that taxonomies do not consider relationships others than "`has a`" or "`is a`". On the contrary, the proposal stated by the AMIGO project, using a declarative language for semantic service specification, accomplish a thorough description of the context and all the elements and services involved in it. However, the semantic model provided, for being so specific, does not suffice to support automatic service composition, but a simple service aggregation, that is, there is no new functionality inferred from existing services.
The main strength of the semantic model, here proposed, lays in its simplicity. This simplicity eases the process of semantically describe any element of the context, without missing any detail. Furthermore, in order to not to miss these details, the `specification` concept is in-

troduced to the model, which encompass all the particularities of services, objects, and devices of an specific domain, adopting the shape of properties.

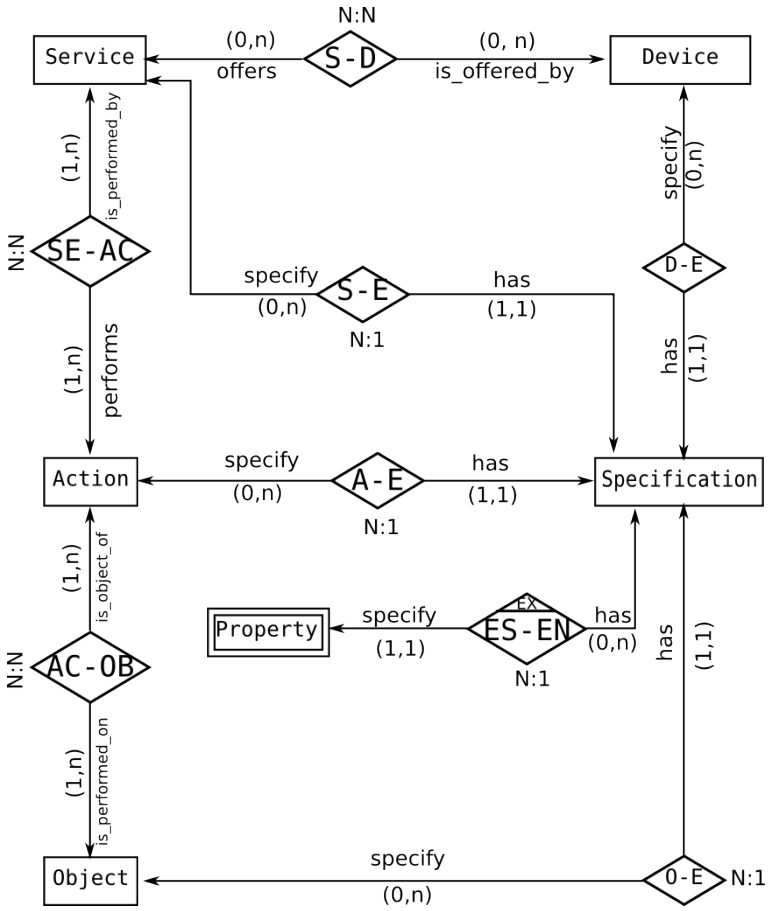

Fig. 2. Entity-Relationship Model for an Ambient Intelligent system.

3.2 The Middleware framework

The role played by the middleware architecture is essential in simplifying and abstracting the complexity and heterogeneity of both, network and device technologies. However, middlewares for traditional environments made some simple assumptions, such as high bandwidth, reliable connectivity, fixed network topology or hardware capabilities, that do not correspond themselves to the reality of an Ambient Intelligence environment.

Middlewares deployed in Ambient Intelligence contexts are used to deal with data generally based on physical parameters, such as temperature, distance, or position. Most of the times, this information is retrieved from sensors, located all along the environment, connected by means of Wireless Sensor Networks (WSN). These sensors are usually hidden so as to keep users unaware of their presence. At the same time, many heterogeneous devices might act as

consumers of this data, such as X10 or EIB, or some others for multimedia support such as HAVi or MHP. Moreover, it is a middleware role to provide common services such as a basic service discovery, event management, resource management, etc. All these requirements, imposed by the Ambient Intelligence context peculiarities, demand a fully equipped middleware framework capable of integrating heterogeneous devices in a transparent way, and transmit them the adopted semantic model as a way of homogenising the interaction with these services and devices.

This work proposes a combination of intelligent agents and a reasoning engine in order to handle the dynamism, ambiguity, and uncertainty of Ambient Intelligence. It should be noticed that the middleware framework is an essential component of the system, since it provides the groundings for the intelligent agents.

Therefore, from a layered perspective, the architecture proposed here rests on top of a powerful middleware framework, that provides the upper layers with the structure, tools and services required to successfully accomplish their tasks. A deep description of the middleware framework details is out of the scope of this work, nonetheless, the most relevant issues are detailed underneath.

This middleware framework, known as DOBS (Distributed Object Based Services), takes the form of distributed object based services. Figure 3 depicts the key components and services of the framework at the core of its great potential.

The **DOBS interfaces** basically standardise the way how services are modelled and con-

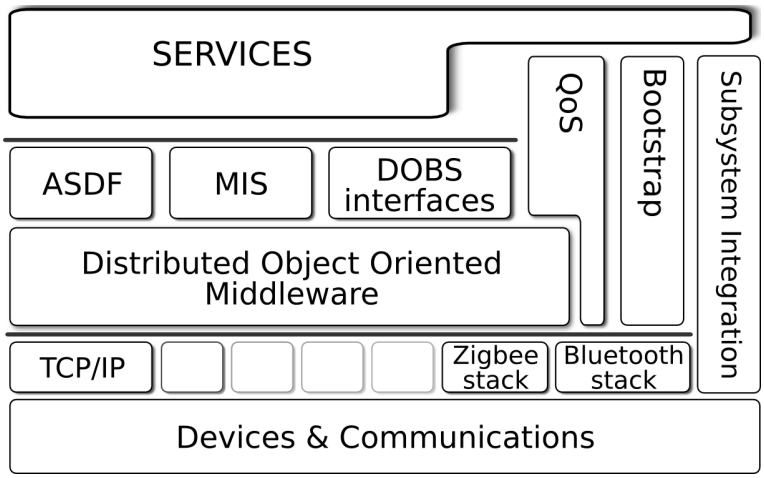

Fig. 3. DOBS framework overview.

trolled. The fact that these interfaces are derived from the semantic model makes them well known and common to all services and modules of the considered architecture. Therefore, it simplifies the way how services are accessed and controlled.

Apart from a common set of interfaces the middleware framework also provides a set of commonly used functionalities, such as the *service discovery, bootstrap service, security mechanisms*, and so forth. **Integration subsystems** work as technological bridges, allowing a seamless integration of services from different technologies. So far, subsystems for UPnP, X10, and Bluetooth service integration are currently available.

Finally, the **model information system (MIS)**, based on the aforementioned semantic model, is intended to provided a common nomenclature to services that eases the understanding and reasoning task carried out in the upper layers.

4. Intelligent Agents Supporting Automated Service Composition

The underlying idea behind the work here described, is to provide the appropriate combination of technologies and the right semantic model, capable of allowing systems for Ambient Intelligence contexts to exhibit an autonomous behaviour, driven by a set of objectives that are to be achieved, satisfied, or maintained. Furthermore, these systems are also expected to be proactive and to foresee curses of actions that lead them to the expected situations or objectives. If necessary, these systems resort to other systems, also deployed in the context, establishing a collaboration or cooperation pattern that leads to the final objectives.
In the seeking of the best approach to satisfy these requirements, an agent-based solution turns out to be the most compelling mean to this end. The motivation behind this choice is twofold. Firstly, given the service oriented character of the middleware framework, the agent-based approach can be easily fit into the framework, adopting the shape of yet another middleware service. Secondly, autonomy and pro-activity features are inherent to agents. In addition to this, the BDI model of agency provides the goal-oriented character, required by the architecture here proposed. However, an agent based approach needs to be enacted with some other low level capabilities, that provides the agent system with a transparent way of communicating with other elements of the context, and controlling the sensors and actuators of the context, as well as an homogeneous access method to all the services deployed in the context. These are too specific capabilities that are generally overlooked by agent frameworks, that delegate the responsibility of providing such capabilities to the middleware framework. Entrusting these low level capabilities to the middleware layer, allows the multi-agent system to remain unaware of the implementation details, giving rise to a modular design.
Nevertheless, the middleware platform does not suffice to cope with the uncertainty, ambiguity, and imprecision of the context information managed by Ambient Intelligence systems. These features state a new demand, as it is the understanding of what is happening in the context, so that this contextual information can be enclosed and get rid of the ambiguity. In this regard, a reasoning engine is revealed as the key element of the overall architecture, since it makes use of the information gathered and endows it with the semantic meaning that supports the reasoning process. Among the many different approaches supporting the reasoning process, for its simplicity and easy integration, this work resorts to a rule-based reasoning engine, known as CLIPS.
The following subsections undertake a thorough description of the agent layer, and the role it plays in the overall architecture. Some implementations details are offered so as to understand how the combination of the multi-agent system, the middleware layer, and the reasoning engine manage to deal with the arisen requirements of an Ambient Intelligence context.

4.1 The Multi-Agent Service Composer System (MASCS)

The proposed multi-agent system solution consists of three intelligent agents: the *Manager*, the *Retriever*, and the *Actor*, described in terms of their beliefs, desires and intentions, as BDI agents. However, since a JADEX implementation is used, this information is translated into beliefs, plans and goals, proper for each application context.
The *Manager* agent plays the role of the context supervisor, and to this end, it strongly depends

on interactions with the middleware platform, so as to communicate with services in a transparent way. The sort of interactions established with the middleware platform are mainly based on an event channel approach.
Services are characterised by the sort of *interface* they implement. Basically, services can be *active services* when they publish their state whenever a change occurs. Sensor services are a good example of active services. The other type of services are the *reactive services* which are capable of answering to the requests published in an event channel. Therefore, considering that services are univocally identified by means of the proxy concept (ZeroC, 2008), and the fact that knowing the implemented interface supposes an univocal way of identifying the methods that can be invoked over the services.
The manager agent is also committed to assure a minimum level of system functionality, overcoming the lack of services, service failures or disappearance. In this endeavour, the manager agent has a set of context goals that are to be maintained or achieved. Whenever the context conditions fail to fulfil the ideal state, the manager agent gets engaged on returning to the *ideal* state. First step consists of launching the *Retriever* and the *Actor* agent, that get noticed of the cause that lead to this undesirable situation.
The retriever agent is basically in charge of gathering the appropriate context information, used in the reasoning process. Therefore, the semantic model and the context information have to be translated into rules, although the difference between using a XML language or the CLIPS language is minimum. Finally, the context objectives or intentions are expressed as rules, stating a set of actions, that according to the current state are capable of leading the context to the envisage state.
The selected actions are carried out by the *Actor* agent, which is in charge of translating into method invocations the semantic action inferred by the rule-based reasoning engine.
Sometimes the inferred semantic actions involve some sort of composition, since there are no basic services capable of providing such functionality. Therefore, when a service composition is required in order to perform the inferred semantic action, the actor agents takes the role of a *planner*, and using an approach based on the hierarchical task network (HTN)(Amigoni et al., 2005), it manages to generate a plan that leads to the composite service. The plan is composed of basic actions that are to be performed over specific objects.

4.2 Integration of the Reasoning Engine

The domain specific knowledge also needs to be provided to the architecture so as to capture the peculiarities of the different contexts where the architecture can work. Once again this knowledge needs to be translated into OWL and combined with the semantic model describing the architecture. Despite its great relevance, this knowledge does not suffice to support the reasoning and inference task on itself, since the reasoning capabilities of the intelligent agents are mainly constrained to their plans and goals, having to resort to an external reasoning engine to achieve broader reasoning capabilities.
Among the different approaches supporting the reasoning task, such as those based on domain logic, ontologies, or declarative languages, an approach based on the combination of a semantic model and a rule-based system that adopt the same semantic model, largely bears the context reasoning and understanding demanded by the composition task. Provided with this knowledge, a rule-based reasoning engine holds enough information to understand the capabilities of the services, as well as to infer new capabilities out of the raw ones.
The ontology classes are mapped into CLIPS classes, while the relationships of the semantic

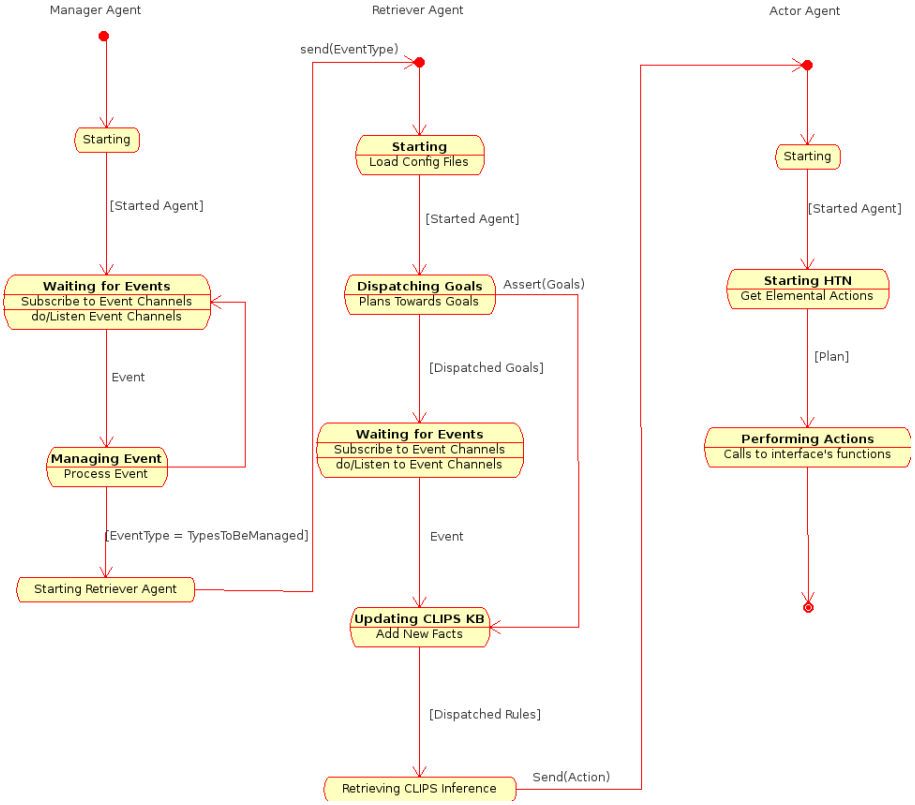

Fig. 4. State Diagram for the Multi-Agent System

model are managed as properties in the ontology, and slots of the CLIPS classes. For instance, the following listing confronts the OWL and the CLIPS definitions for the *action* entity.

Listing 1. OWL code for the *action* class

```
<owl:Class rdf:about="#Action">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/
        >
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:someValuesFrom rdf:resource="#Specification"/>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#has-a-specification"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
```

```
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:someValuesFrom rdf:resource="#Object"/>
          <owl:onProperty>
            <owl:ObjectProperty rdf:ID="action-is-performed-on-object"/>
          </owl:onProperty>
        </owl:Restriction>
      </rdfs:subClassOf>
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#is-part-of-service"/>
          </owl:onProperty>
          <owl:someValuesFrom rdf:resource="#Service"/>
        </owl:Restriction>
      </rdfs:subClassOf>
    </owl:Class>
```

Listing 2. CLIPS code for the *action* class

```
        (defclass ACTION
          (is-a USER)
          ;(create-accessor write)
          (slot ID
                  (type SYMBOL)
                  (allowed-symbols action-$?))
          (slot has-a-specification
                  (allowed-symbols specification-$?)
                  (default ?NONE))
          (multislot action-is-performed-on-object
                  (allowed-symbols object-$?)
                  (default ?NONE))
          (multislot is-part-of-service
                  (allowed-symbols service-$?)
                  (default ?NONE)))
```

The Protégé tool provides an utility to automatically generates the CLIPS code out of the RDF/XML ontology description, what simplifies the migration of the semantic model to the reasoning engine.
Regarding the implementation details, CLIPS provides an extensive API in C language. Nevertheless, it is easily used from the agents plans, written in Java, thanks to the Java Native Interface framework, also provided by CLIPS. Therefore, the integration of the reasoning system with the intelligent agent platform is almost straightforward, and can be reduced to call to the API functions.

5. A case study

Surveillance contexts are one of those fields where Ambient Intelligence systems can be of great help. Nevertheless, this is far from being a reality, and at the moment, these systems require human supervision when making important decisions or when wise reactions are expected from them. An ideal Ambient Intelligence system, instead of being constrained to a

fixed set of possible reactions, it is supposed to actively anticipate and make decisions under whatever the circumstances. This section draws on a simplified case study for depicting the important advances that could be achieved by implementing the proposal here presented.
The surveillance context considered for this case study is constrained to a building. It counts on several devices, such as cameras, presence, noise and light sensors, among some. These devices are placed all around the building, with the intention of supervising the activity of the critical areas. This case study considers a restricted access corridor, where an alarm arises as a result of an unauthorised presence. Along this corridor, three presence sensors can be encountered, as well as two cameras focusing the entrance of the rooms, located at the two ends of the corridor.
Founded on figure 4 that describes the state diagram for the multi-agent system, the first step accomplished by the manager agent, once started, consists on subscribing to certain event channels, where services publish their state and answer to requests. This is to say that, when the presence sensor changes its state to *activated*, as result of a person breaking into a restricted access area, the sensor service sends a message to the event channel, publishing its new state. This message is then received by the manager agent, that interprets this message as an *unauthorised presence alarm*, and gets into the state of dealing with an *unauthorised presence state*. The manager starts and notifies the retriever agent, which afterwards establishes the new context objectives, as well as the plans to achieve them, as it can be seen in the following code, relating to the retriever agent description, using the JADEX nomenclature.

Listing 3. Some of the most relevant goals and plans for the *Retriever* agent

```
1
2     <!--
3         *********************************
4         ******** List of goals **********
5         *********************************
6       -->
7
8       <!-- O1. Find out the intrusion intention -->
9       <achievegoal name="discover_intentions">
10        <creationcondition>$beliefbase.event.isTYPEUNAUTHORISEDPRESENCE()<
              /creationcondition>
11        <contextcondition>$beliefbase.intruders.isSTATEONRUNNING()</
              contextcondition>
12        <targetcondition>$beliefbase.intruders.getIntruderHasAimObjective
              ()!=null</targetcondition>
13      </achievegoal>
14
15      <!-- O2. Intruder identification -->
16      <achievegoal name="intruder_identification">
17        <creationcondition>$beliefbase.event.isTYPEUNAUTHORISEDPRESENCE()<
              /creationcondition>
18        <contextcondition>$beliefbase.intruders.getIdentification.equals(
              null)</contextcondition>
19        <targetcondition>$beliefbase.intruders.getIdentification()!=null</
              targetcondition>
20      </achievegoal>
21
22      <!-- O3. Damages caused during intrusion -->
23      <maintaingoal name="caused_damages" recur="true"
```

```
                    recurdelay="2000">
      <creationcondition>$beliefbase.event.isTYPEUNAUTHORISEDPRESENCE()</
          creationcondition>
      <maintaincondition>$beliefbase.intruders.isSTATEONRUNNING()</
          maintaincondition>
    </maintaingoal>

    <!-- O4. Stop the intruder -->
    <performgoal name="stop_intruder" retry="true" exclude="never">
      <creationcondition>$beliefbase.event.isTYPEUNAUTHORISEDPRESENCE()</
          creationcondition>
      <contextcondition>$beliefbase.intruders.isSTATEONRUNNING()</
          contextcondition>
    </performgoal>

    <!-- O5. Mantain the safety of the environment -->
    <performgoal name="keep_safe" retry="true" exclude="when_failed">
      <parameter name="intruder" class="Intruder">
        <bindingoptions>$beliefbase.intruders</bindingoptions>
      </parameter>
      <unique/>
      <!-- Create a new goal when new intruder is seen and
           the agent isnt already keeping environment safe. -->
      <creationcondition>$beliefbase.event.isTYPEUNAUTHORISEDPRESENCE()</
          creationcondition>
      <!-- Suspend the goal when the intruder has been caught. -->
      <contextcondition>$beliefbase.intruders.isSTATEONRUNNING()</
          contextcondition>
      <!-- The goal will be dropped when the intruder has vanished. -->
      <dropcondition>$beliefbase.intruder==null</dropcondition>
    </performgoal>
    <achievegoalref name="df_deregister">
      <concrete ref="dfcap.df_deregister"/>
    </achievegoalref>

  </goals>


  <!--
      ********************************
      ******** List of plans **********
      ********************************
    -->
  <plans>
    <!-- Plan to get the route followed by the intruder -->
    <plan name="get_route">
      <body class="GetRoutePlan"/>
      <trigger>
        <goal ref="discover_intentions"/>
        <goal ref="stop_intruder"/>
      </trigger>
    </plan>

    <!-- Plan intended to obtain likely vulnerabilities -->
    <plan name="get_vulnerability">
```

```
73          <body class="GetVulnerabilityPlan"/>
74          <trigger>
75            <goal ref="discover_intentions"/>
76          </trigger>
77        </plan>
78
79       <!-- Plan intended to obtain images from the intruders -->
80        <plan name="get_snapshot">
81          <body class="GetSnapshotPlan"/>
82          <trigger>
83            <goal ref="intruder_identification"/>
84          </trigger>
85        </plan>
86
87       <!-- Plan intended to sucess in the intruder face recognition -->
88        <plan name="face_recognition">
89          <body class="FaceRecognitionPlan"/>
90          <trigger>
91            <goal ref="intruder_identification"/>
92          </trigger>
93        </plan>
94
95        <!-- Plan intended to gathered the damages -->
96        <plan name="get_damage">
97          <body class="GetDamagePlan"/>
98          <trigger>
99            <goal ref="caused_damages"/>
100         </trigger>
101       </plan>
102     </plans>
```

Among the goals driving the agent behaviour, the first one is intended to discover the intruder aim, thought of as the building place where the intruder is going to. The plans provided to achieve the goal of `discoverintentions` are the ones referred as `GetRoutePlan` and `GetVulnerabilityPlan`. The first plan is meant to obtain the route that the intruder is following, as an attempt to provide the guards with this route information so that intruders can be more easily caught up. The second plan seeks for keeping an updated list of the likely objectives, in such a way that depending on the relative importance of an objective, and the proximity of the intruder, this plan provides the guards with an ordered list of the likely objectives of the intrusion.

Here the domain specific knowledge, modeled using an E-R diagram, as depicted in figure 5 is combined with the semantic model proposed for the system and depicted in figure 2. In order to provide the reasoning system with all this semantics, concepts and their relationships of the domain specific knowledge are coded as CLIPS classes, and slots.

Therefore, the context objectives that are likely to be a target of intrusion attacks are mapped into the `Objective` CLIPS classes, and ranked according to their subjective importance. Here, the retriever agent is constantly asserting the feedbacks of the intruder position, using to this end the information gathered from the contextual sensors and services. The faculty to gather or request the appropriate services in order to keep track of the intruder is founded on the ability to distinguish those services that implement the action of *detect* or *sense* `persons` or *objects*. Since all the services deployed in the system are described in terms of the actions

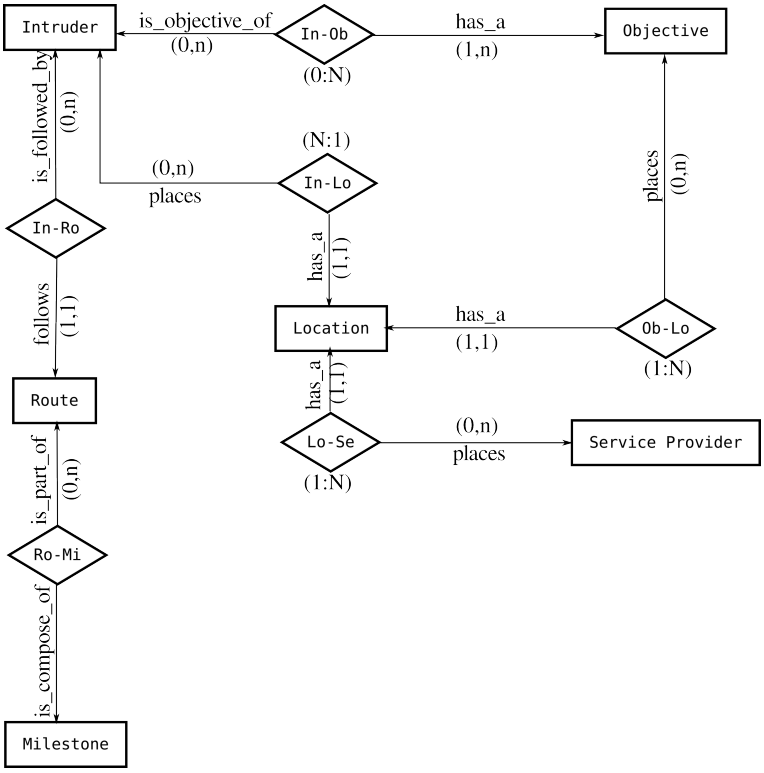

Fig. 5. Entity-Relationship Diagram for the Unauthorised Presence State Domain

they can accomplish and the objects on which these actions are preformed, the retriever agent launches the plan in charge of retrieving updated information of the intruder location, by subscribing to all those channels where services deployed at the current intruder location publish their state.

The plan intended to discover the route followed by the intruder shows a particular case of service composition, rather than a simple service combination that basically joins service functionalities. This is the case of a video camera service, used as a presence sensor, if combined with a service of face detection, in such a way that if a face is detected, presence can be inferred.

Selecting those services that at a certain location provide specific functionalities, as for this example would be *detecting people*, suffices to the actor agent to make some inquiries about these services state, either by subscribing to the appropriate channels or by direct method invocation, since the implemented interfaces are determined by the type of service.

```
1  (defmessage-handler INTRUDER get-location()
2          (do-for-all-instances ((?serv SERVICE))
3          (if (eq (send ?serv in-the-same-area ?self:x ?self:y) TRUE) then
4          (bind ?action ?serv:service-consits-of-action)
5          (bind ?object ?serv:service-consits-of-object)
```

```
6           (if (> (str-index ?action detect) 0) then
7                   (if (> (str-index ?object person) 0) then
8                           (bind ?selected-service ?serv)))
9           (printout t "The service "  ?serv:ID "detects presence in the
                area where the intruder was last located"))
```

6. Conclusions and future works

This work has presented a novel approach towards Ambient Intelligence, mainly supported on the benefits derived from the Automated Service Composition, achieved by means of a multi-disciplinary approach, that combines an omnipresent semantic model with a middleware platform and a reasoning engine, orchestrated all this by a set of intelligent agents. As constituent components of a broader architecture, these are integrated in a middleware platform that provides them with the groundings to support their endeavours towards intelligent environments.

Nevertheless, this proposal does not represent a silver bullet to achieving self-managed systems, since the main drawback encountered when trying to develop systems for Ambient Intelligent lies on the vast amount of domain specific knowledge required when supporting systems for Ambient Intelligence contexts. Despite the availability of reasoning tools capable of dealing with domain knowledge, they reveal futile without the common sense knowledge support. However, this approach claims to solve many of the more common shortcomings of approaches intended to cope with Ambient Intelligence. Further research efforts are to be dedicated to the automation the of workflow generation just as accomplishing an extensive domain knowledge modelling by means of a common-sense framework as OpenCyc (Cycorp, 2008a)(Cycorp, 2008b).

7. Acknowledgments

This work has been funded by the European Regional Development Fund, the Regional Government of Castilla-La Mancha, the Spanish Ministry of Science and Innovation, and the Spanish Ministry of Industry under grants PAI08-0234-8083 (RGrid), TEC2008-06553 (DAMA), and CENIT Hesperia.

8. References

Amigoni, F., Gatti, N., Pinciroli, C. & Roveri, M. (2005). What planner for ambient intelligence applications?, *IEEE Transactions on Systems, Man, and Cybernetics, Part A* **35**(1): 7–21.

Bratman, M. E. (1987). *Intention, Plans, and Practical Reason*, Harvard University Press, Cambridge, MA.

Costa, P., Coulson, G., Mascolo, C., Mottola, L., Picco, G. P. & Zachariadis, S. (2007). A reconfigurable component-based middleware for networked embedded systems, *Int. Journal of Wireless Information Networks* **14**(2).

Cugola, G. & Picco, G. P. (2006). Reds: a reconfigurable dispatching system, *SEM '06: Proceedings of the 6th international workshop on Software engineering and middleware*, ACM, New York, NY, USA, pp. 9–16.

Cycorp, I. (2008a). The cyc project home page. Available online at: `http://www.cyc.com`. Retrieved on December 10th, 2008.

Cycorp, I. (2008b). The opencyc project home page. Available online at: `http://www.opencyc.org`. Retrieved on December 10th, 2008.

Ducatel, K., Bogdanowicz, M., Scapolo, F., Leijten, J. & Burgelman, J. C. (2001). Istag: Scenarios for ambient intelligence in 2010, *Technical report*, ISTAG.

Issarny, V., Sacchetti, D., Tartanoglu, F., Sailhan, F., Chibout, R., Levy, N. & Talamona, A. (2005). Developing ambient intelligence systems: A solution based on web services, *Automated Software Engg.* **12**(1): 101–137.

Pokahr, A., Braubach, L. & Lamersdorf, W. (2005). Jadex: A bdi reasoning engine, *in* J. D. R. Bordini, M. Dastani & A. E. F. Seghrouchni (eds), *Multi-Agent Programming*, Springer Science+Business Media Inc., USA, pp. 149–174. Book chapter.

Prete, L. D. & Capra, L. (2008). Mosca: seamless execution of mobile composite services, *ARM '08: Proceedings of the 7th workshop on Reflective and adaptive middleware*, ACM, New York, NY, USA, pp. 5–10.

Rao, A. S. & Georgeff, M. P. (1991). Modeling rational agents within a BDI-architecture, *in* J. Allen, R. Fikes & E. Sandewall (eds), *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, pp. 473–484.

Weiser, M. (1995). The computer for the 21st century, pp. 933–940.

Wooldridge, M. J. (2000). *Reasoning about Rational Agents*, The MIT Press, Cambridge, Massachusetts.

ZeroC, I. (2008). Ice home page. Available online at: `http://www.zeroc.com/`. Retrieved December 20th, 2008.