Pervasive and Mobile Computing 8 (2012) 562–574

Contents lists available at SciVerse ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

Ubiquitous Agents for Ambient Ecologies

Nikolaos Dipsis [*], Kostas Stathis

Department of Computer Science, Royal Holloway, University of London, UK

A R T I C L E I N F O

Article history:
Available online 4 November 2011

Keywords:
Ubiquitous Agents
Avatars
Virtual Agents
Multi-agent systems

A B S T R A C T

The cost of state-of-the-art devices and robots is critical for the uptake of Ambient Intelligence (AmI). One way to utilize low-cost hardware for both devices and robots is to run smart software remotely as agents deployed on computationally rich environments. In this context, the devices and the robots can be seen as the Avatars of agents, while the way devices and agents are related may be considered as an Ambient Ecology. We show how the application of a middleware platform called EVATAR realizes this vision by exemplifying the main issues with a multi-robot and a smart-home scenarios.

© 2011 Elsevier B.V. All rights reserved.

1. Introduction

Ambient Ecologies can be construed as a set of interconnected objects in the form of devices and robots that are distributed in a finite physical environment and related between them with the intention to assist humans to achieve their everyday tasks. The word "ecology" is used as a metaphor to emphasize the idea of a set of devices being related to one another. We can even speak of devices in symbiotic relationships, where certain types of devices are "helping" others in achieving their goals, thus forming a device ecology consisting of devices with different forms, functionalities and sizes (Seera et al. in [1]).

This work focuses on making devices, whether in the form of robots or otherwise, smarter so that they become more aware of one another, of their capabilities and of their environment. In other words, we want devices to become more aware of the context in which they are embedded. Furthermore, we want to aggregate the functionality of simple devices (for example smart home sensors and actuators) to create more sophisticated devices or make them cooperate. As a result, the answer to the question of how to achieve the aggregation of functionality is very important, especially if we aim to develop practical applications.

Many commercial household robots and AmI devices are typically constrained by limited computational and connectivity capabilities of hardware and operating systems. This impedes the interoperability among heterogeneous devices within an Ambient Ecology and as a result any attempt to aggregate these devices into a system that tries to achieve a common goal. Although state-of-the-art hardware and software with all the characteristics required for the development of such intelligent systems are already available for demonstration purposes, the cost is inhibiting commercial, household systems from exhibiting similar behaviors.

Several middleware and development frameworks have been proposed to integrate hardware and software components to AmI ecologies such as those in [2–7]. These frameworks predominantly distribute the computation and system resources among interconnected devices for integrating heterogeneous components while focusing in defining clear well-documented interfaces permitting the reuse of existing code. Although they generally reduce the difficulty and cost associated with prototyping such applications they are typically constrained by the limited computational capabilities of robotic/smart device hardware and operating systems.

[*] Corresponding author.
E-mail addresses: nikolaos@cs.rhul.ac.uk (N. Dipsis), kostas@cs.rhul.ac.uk (K. Stathis).

1574-1192/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.pmcj.2011.10.006

To tackle the above problems we use the notion of *Ubiquitous Agent (UA)* as an entity that is capable of simultaneous presence in both a virtual and a physical environment. We build UAs, through enabling Virtual Agents (VAs), software agents residing in virtual networked environments, to connect to and control the aggregation of one or more devices in the physical world. At the same time they may still bare virtual sensors and actuators allowing them to interact with their virtual environment. In the rest of the paper, we will further use the term *Avatar* to represent a mobile robot or any aggregation of connected devices (whether sensors, actuators or smart devices) that are embedded in a finite physical environment such as a smart home. Put another way, a UA $=$ VA $+$ Avatar.

We will assume that VAs usually run on powerful, in terms of computational capabilities and connectivity, MAS (Multi-Agent Systems) platforms. Thus in our approach the computationally expensive reasoning, planning, perception and communication are performed by the VAs in the MAS platform while the acting and the sensing in the physical world through the corresponding Avatars of these VAs. Also, enabling a VA to control a variety of smart devices implies that the Avatar for that VA will aggregate these devices into a more sophisticated component, further ensuring interoperability.

Our hypothesis then is that it is possible to wirelessly connect an artificially intelligent VA residing in a multi-agent system that is rich in computational resources with a set of devices that have limited computational capabilities, in order to aggregate the devices into a system that will be controlled by the VA as an Avatar. Our aim is to develop a software architecture and framework to build UAs in terms of VAs and Avatars in the context of Ambient Ecologies. More specifically, our objective is to develop a software platform whose components will facilitate the binding of artificially intelligent VAs to their Avatars and enable the graceful communication and interaction.

With regards to the VA architectures considered by our approach, as MAS platforms vary in terms of how perception and action execution of an agent is implemented, we assume that there will be a straightforward correspondence between virtual perception/action execution and the sensing/actuation of the Avatar. Indeed, agent platforms like GOLEM [8] and CaRTAGo [9] use virtual sensors and actuators explicitly as components of a Virtual Agent's body, while platforms like JADE [10] use it implicitly as part of sending and receiving messages.

We study a framework that realizes the above ideas in order to create distributed architectures that enable VAs to achieve ubiquity via Avatars in the physical world. Our framework consists of BIL (Body Integration Language), a middleware called EVATAR (Embodying Virtual Agents Toward Autonomous Robots) and a distributed event-based architecture that allows the creation of UAs. An early version of EVATAR was presented in [11]. In this article we discuss in detail the latest version of EVATAR, featuring an improved methodology for application to UAs with one instance of the middleware per agent. We describe the connections and the new architectures it supports and we deal with resilience in EVATAR based applications.

The next section presents BIL, EVATAR and how to integrate VAs and Avatars into a distributed architecture. Section 3 describes two example scenarios. In the first example scenario, we connect GOLEM [8] VAs to LEGO robot Avatars. Despite the computational, memory and in general resource limitations of the robots, they perform collaborative behaviors in the physical world taking advantage of the planning, reasoning and connectivity capabilities of agents following "Observe-Choose-Act" [12] cycles. In the second example scenario, we connect GOLEM VAs to a collection of low-cost sensors, actuators and devices in order to support activity recognition for a "Smart Home" security scenario. In Section 4 we discuss and compare our framework based on EVATAR to existing approaches in the literature. We conclude in Section 5 where we also present our plans for future work.

2. A framework for Ubiquitous Agents

This section provides an overview and a high-level picture of Ambient Ecologies developed using UAs. We present the architecture of a UA and we provide a conceptual description of the UA components and how they are linked using the EVATAR middleware. The detailed architecture of individual components is also discussed in the context of the software pattern used to implement the EVATAR system. We also discuss how to achieve resilience within an Ambient Ecology when there is poor connectivity or faulty behavior of an EVATAR instance.

2.1. Overview

A typical system that uses EVATAR (Fig. 1) consists of three elements that are interconnected within a network (a local network when execution speed is critical or even the internet if it is not): VAs in a MAS, instances of the EVATAR middleware and Avatars. As we can see these elements belong to conceptually different worlds, the VAs in the virtual world, their Avatars in the physical world and EVATAR in between. There is always an one-to-one relationship between a VA, an instance of the EVATAR middleware and an Avatar. There are multiple instances of the EVATAR middleware running in the networked system of Fig. 1. An instance may run on the same computer as the virtual MAS or distributed on a network of computers, as long as there is connectivity between the middleware and the particular VA and Avatar of the one-to-one relationship.

EVATAR is designed to implement a Service-Oriented Architecture [13]. Every link between VA and Avatar components is essentially a two-way service provider–consumer relationship. The architecture assumes BIL, which is essentially a metadata language allowing the descriptions of the expected services providing sufficient detail to describe their characteristics and data. Also, it allows for the description of communication protocols by defining signals for Avatar control or data passing. The EVATAR middleware uses BIL in order to enable the implementation of the architecture. It connects services and consumers aggregating them into a distributed solution dealing with runtime messaging, message design and inter-service controls.

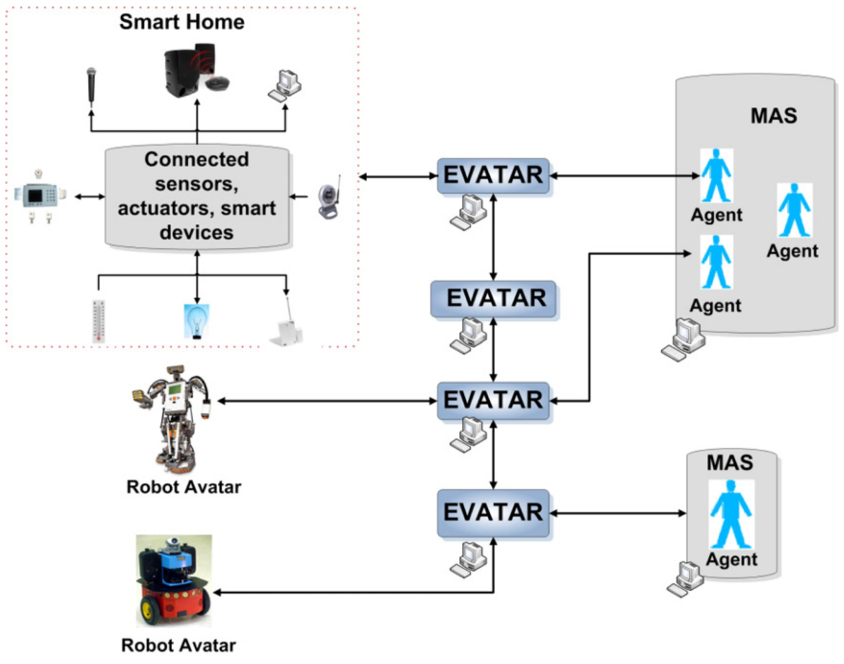

Fig. 1. Overview of a distributed system using EVATAR.

2.2. *Ubiquitous Agent architecture*

A UA features virtual sensors and actuators of a VA to interact with a virtual environment as well as "augmented" sensors and actuators to interact with the physical world. An augmented sensor in our architecture is a virtual sensor that is capable of receiving sensory data from: (a) the control software of a physical sensor that provides raw data e.g. the luminosity levels of a light sensor (b) a high-level software that processes the input of a single or of multiple physical sensors and transforms it into a high-level property of an observable quantity that will be used as the input to the agent. For example we could have a camera sensor and a motion detection software that, combined, would constitute a motion sensor providing the agent with an input using the keywords "motion detected". Similarly an augmented actuator is a virtual actuator that is extended to control: (a) the control software of a physical actuator, sending low level control signals (b) a high level software that receives an abstract command from the VA and utilizes a single or multiple physical actuators to perform an action in the real world. For simplicity purposes, we will be calling physical sensors and physical actuators both of the types of sensors and actuators that we described above. EVATAR typically treats smart devices as actuators.

Fig. 2 illustrates an overview of our approach portraying a UA bearing both virtual and augmented sensors and actuators allowing it to interact with both virtual and physical environments. Virtual sensor or actuator data is generally different in nature and format to the heterogeneous physical actuator software or the physical sensor data. To create augmented actuators we need to be able to transform virtual actuator control data into a format that is suitable to control the corresponding physical actuator. Similarly a transformation is required when virtual sensors acquire sensory data from physical ones. To achieve the above we use special objects within the MAS environment that we call sensor and actuator emulators. They are MAS objects emulating the functionality of physical sensors and actuators in a way that the virtual ones do not interact directly with them; instead they interact with the emulator objects.

From the virtual sensor perspective these objects transform and emit the sensory input of the physical sensor within the virtual environment. Essentially, they internalize within the MAS the stream of sensory data derived from the physical sensor by transforming it into a suitable data abstraction for the corresponding virtual sensor. For example a virtual light sensor will be listening to the output of a luminosity emulator object that provides an abstraction of the physical luminosity sensor instead of listening to the physical sensor itself. Conceptually, a virtual sensor is using the sensor emulator as a "window" to the physical world the same way as the human eye is using a computer screen as a window to a virtual reality game.

From the actuator perspective, a virtual actuator will typically interact with the corresponding actuator emulator object, which in its turn will be transforming this interaction into events that will control a physical actuator. It is similar to a gamer waving a Nintendo Wii controller [14] to instigate the waving of the arm of an Avatar in a virtual computer game. The emulator is the "controller" of the VA controlling an actuator (Avatar arm) in the physical world.

A key feature of the emulator objects is that they are implemented as part of an extended agent design (Fig. 2) and they allow the agents to connect to any physical sensor or actuator that matches the emulator abstraction. For the remainder of this paper, when we mention a virtual sensor or actuator in the context of a connection to a sensor or actuator of an Avatar, we will mean an extended virtual sensor\actuator with an emulator object.

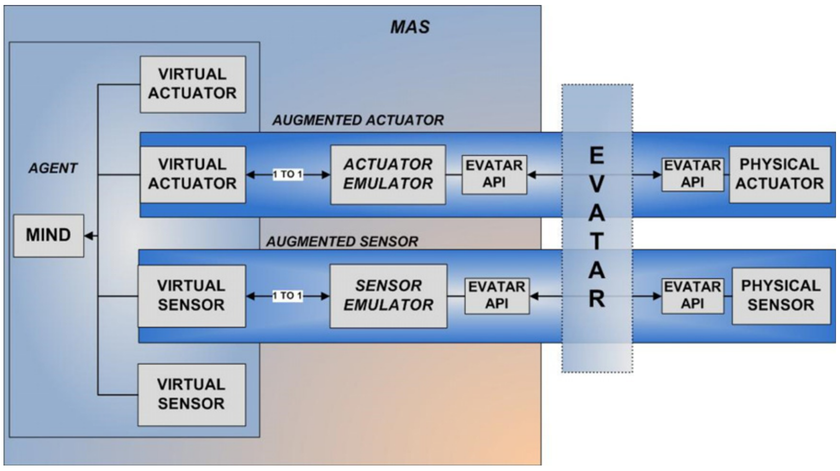

Fig. 2. A Ubiquitous Agent whose VA is shown to have two virtual sensors and two virtual actuators with a virtual sensor and actuator connected to an Avatar's actuator and sensor via EVATAR.

To create a distributed application consisting of the VAs, collections of sensors and actuators embedded in "Smart Homes" or mobile robots and their connections, we need a middleware such as EVATAR. We attach the EVATAR API (interface to EVATAR) on the sensor and actuator emulators and the control software of the physical sensors and actuators in order to enable EVATAR to route and coordinate the exchange of messages between them thus allowing the creation of the connections. At this point we need to provide a background on EVATAR to illustrate how we implement the above.

2.3. *EVATAR middleware*

Essentially, we implement a service-oriented architecture in which every sensor and actuator of the agents and the Avatars is implemented to present service provider or service consumer functionality or in many cases both. The architecture allows the creation of meaningful provider–consumer relationships. For example, a virtual actuator (extended with an emulator) instead of acting in the virtual environment, it sends a signal to consume the service offered by the physical actuator control software. On the other hand, when a physical sensor sends sensory data to the virtual one, the service offered by the extended with an emulator virtual sensor is to pass the data to the agents reasoning component. In the following we will describe BIL and EVATAR in more detail.

2.3.1. *Body Integration Language*

BIL (Body Integration Language) is an XML based metadata language providing abstract descriptions of VA and Avatar bodies and allowing the definition of communication protocols for the connections between them. As we previously mentioned, virtual sensors and actuators within the context of connections to physical ones are referring to the extended designs that encapsulate the emulators.

BIL describes Virtual Agent actuator control software as a service consumer of the service offered by the corresponding Avatar actuator software. Also, a virtual actuator may provide the service of passing an acknowledgment from the Avatar to the VA reasoning component acting as a BIL service as well. On the other hand, VA sensors are typically described by BIL as services as they offer the service to pass the sensory data that they receive from the Avatar sensor consumers to the agent mind. Similarly they may encapsulate consumer behavior as well as it is possible that they may also request sensory data from the Avatar sensor component. Therefore, VA or Avatar sensors and actuators may be described by BIL as services, service consumers or both at the same time. BIL describes in sufficient detail the characteristics of these services and their data. To describe a service, it uses custom descriptive semantics (example in Fig. 3).

BIL descriptions are similar in concept and in principle to the ones of the Web Services Description Language (WSDL) that is used to describe web-services [15]. BIL uses a different and more minimalistic syntax missing components that are irrelevant to the EVATAR framework. A BIL description consists of the following elements: the "description" (root), the "context" where we describe the context data of the description and the "types" where we describe services and consumers. Fig. 3 shows a BIL description for an actuator component that provides the service of turning a light on and consumes the service of sending feedback back to the VAs reasoning component.

Instead of the services embedding calls to each other in their source code we use BIL to define protocols that describe how services pass. We associate software functionality based on the available services offered by agents and their Avatars, their characteristics, and the means to build an application utilizing these sources. The definition of the communication protocols used by EVATAR is inspired by SOAP [16]. The protocols are implemented by defining BIL signals (messages). These signals are created based on the BIL descriptions of the services. BIL supports several types of signals for the mapping

```
<?xml version="1.0" encoding="utf-8"?>
<bil:description xmlns="HomeLightActuator01">
  <bil:context>
    <EntityID>7008</EntityID>
    <API_NAME>LIGHT_ACTUATOR</API_NAME>
    <EntityType>Avatar</EntityType>
    <Context>ActuatorDescription</Context>
  <bil:context>
  <bil:types>
      <bil:Component>
        <ComponentID>HomeLightActuator01</ComponentID>
         <Service>
         <ServiceCategory>ACTUATOR</ServiceCategory>
          <ServiceType>TurnLightOn</ServiceType>
          <ServiceParameters>
              <Light>yes</Light>
            <Email>no</Email>
            <TextMsg>no</TextMsg>
            <RotationSpeed>no</RotationSpeed>
            <Duration>no</Duration>
            <Angle>no</Angle>
            <Speed>no</Speed>
          </ServiceParameters>
        </Service>
        <Consumes>
          <ServiceCategory>ACTUATOR</ServiceCategory>
          <ServiceType>ACTION_FEEDBACK</ServiceType>
          <ServiceParameters>
            <Feedback>yes</Feedback>
          </ServiceParameters>
        </Consumes>
      </bil:Component>
  </bil:types>
</bil:decription>
```

Fig. 3. BIL description of a light actuator.

and augmentation of Virtual Agent sensors, actuators and behaviors to the Avatars' physical components. Similarly to SOAP the structure of the BIL signal contains: the "Envelope" that encapsulates the header and the "Body". The Body contains elements corresponding to service requests or responses (application specific).

A we can see in the example of Fig. 4, the body of the signal contains information and data for the remote procedure call. BIL signals also include in the header a context field denoting the context of the communication e.g. in our example an agent request awaiting feedback.

2.3.2. The middleware components

To implement EVATAR we need to choose a design pattern that is service oriented. There are several patterns [17] but the big question is which one would optimize the performance of the system while providing with a consistent and robust solution. The main issue that we had to resolve was the fact that the behavior of individual services can vary depending on their design, runtime usage, the workload required to perform a task, the consistency of a service-oriented solution responding to requests and other performance issues.

A similar problem was confronted by Utschig et al. as described in "SOA Design Patterns" by Thomas Erl [17] with the difference that they were dealing with web-services, user interfaces and user requests. Their solution was to implement a user interface Mediator approach. We took this idea on board and created our own version of the Mediator Design Pattern [18] to establish mediator logic ensuring timely interaction and feedback. The mediator pattern is defined as a behavioral design pattern allowing the mediation of networked remote entities (see Fig. 5). Quoting Gamma et al. [18] "The mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently". Our adaptation of the Mediator Pattern facilitates the mediation of the services. The mediator features loose coupling and ensures the interoperability and scalability of an EVATAR application as well as other characteristics such as service abstraction and reusability.

```
<?xml version="1.0"?>
<bil:Envelope xmlns:bil="AgentLightActuator01">
 <bil:Header>
   <EntityID>7007</EntityID>
   <BILTYPE>SIGNAL</BILTYPE>
   <Context>AgentRequestWithFeedback</Context>
 </bil:Header>
 <bil:Body>
   <SignalMSG>SWITCH_LIGHT_ON</SignalMSG>
   <Parameters><Duration>-1</Duration></Parameters>
 </bil:Body>
</bil:Envelope>
```

Fig. 4. BIL signal example for light actuator.

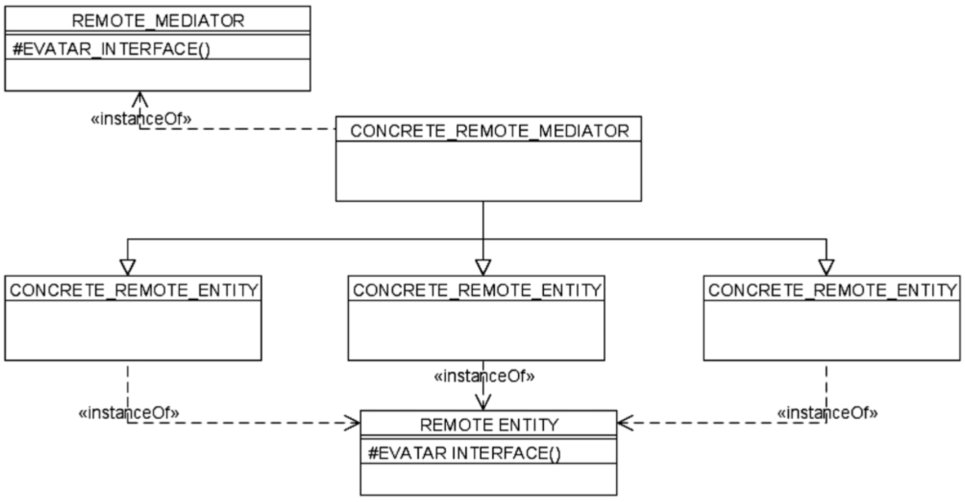

Fig. 5. Mediator.

The EVATAR architecture features two elements: the Remote Entity and the Remote Mediator. We use the term remote entity to denote something that provides or consumes (or both) a service. Within the context of our work it denotes a piece of software that controls what we classified in 2.2 as a sensor or an actuator. The EVATAR API (interface to EVATAR) is the endpoint required from a software entity for participation in the system.

EVATAR acquires the BIL Metadata descriptions from the remote entities through the EVATAR API. When a software entity, which as we defined is a piece of software controlling a sensor or an actuator, joins an EVATAR application it will need to send its BIL description (through its emulator object) within a period of time. EVATAR uses BIL descriptions to register the services and/or the service consumers that they describe in a data structure called the Service Registration List. We call binding the process by which EVATAR autonomously processes the Service Registration List in order to find and store in a separate table compatible service consumer–provider descriptions. EVATAR checks for compatibility by searching and matching keywords in the context of BIL descriptions.

The result of the bindings is the Body Mapping Registry table that is used as a look-up table by EVATAR to identify service–service consumer relationships. It is essentially a data structure holding one-to-one relationships between sender and receiver identifiers that are extracted from the BIL descriptions. The routing and the coordination for the exchange of signals is achieved by EVATAR through matching the sender and receiver identifiers contained in the BIL signals to the relationships of the Body Mapping Registry table. The binding ensures that the correct service will receive a BIL signal with a request for an action.

Despite the fact that EVATAR follows the centralized Mediator approach for the exchange of signals, the actual applications that utilize EVATAR are of distributed nature as we have multiple one to one relationships between VA instances of EVATAR and Avatars distributed in the network. EVATAR coordinates the exchange of signals among the distributed entities but it does not control the applications. The control is assumed by the VAs.

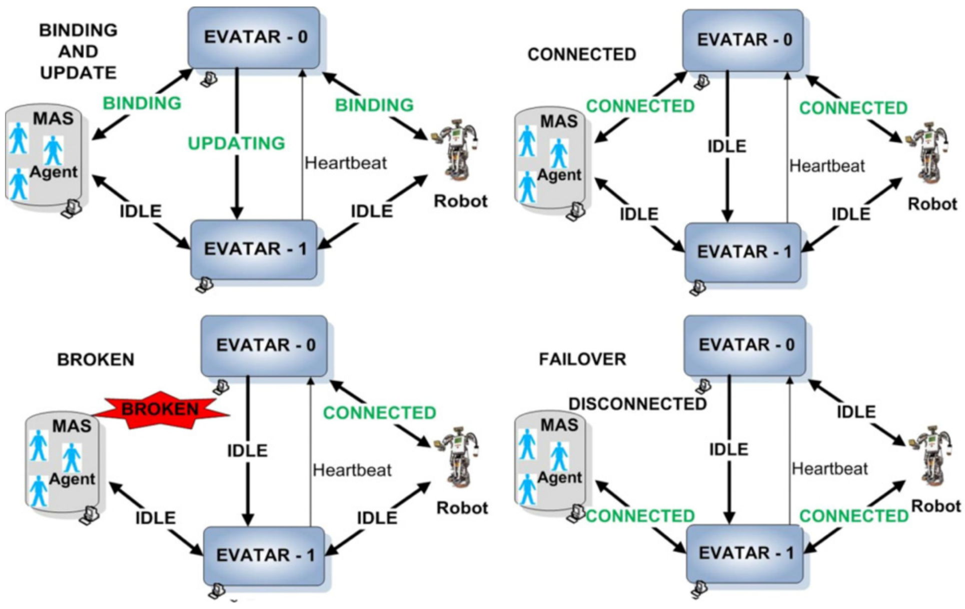

Fig. 6. Failover example in distributed EVATAR.

2.4. Resilience

In a typical EVATAR distributed system (such as the one of Fig. 1), we may see multiple instances of EVATAR supporting one to one relationships between VAs and their Avatars. To achieve resilience in a distributed architecture, EVATAR implements an automated failover approach based on "High-availability clusters" [19]. Specifically, we run instances of EVATAR in multiple hosts (computers, tablets etc.). Some of them (High-availability clustering) are redundant nodes that are used specially for monitoring and failover. As soon as an instance of EVATAR binds a VA to an Avatar, all the relevant data is copied to a redundant EVATAR instance through an update. The redundant EVATAR instance is monitoring a heartbeat on the active instance and as soon as it detects failure, it will be able to take over and become the active instance. Having all the relevant data from the update it is able to continue from the point at which the previous active instance has stopped. Also, it will restart the failed instance of EVATAR that will in its turn become the monitoring instance ensuring a continuity in EVATAR service in case of multiple failures.

In Fig. 6 we see an example of an instance of EVATAR being monitored by a redundant failover EVATAR node running on a different computer. When using high availability clusters, the EVATAR architectures are capable of recovering connectivity and improving the performance of the communications. Each active node is monitoring the performance of its network connections with the VA and Avatar. If a connection is not responsive e.g. if EVATAR sends a message to a VA (e.g. mediating a message from an Avatar body component) and it does not receive acknowledgment within a timeout, then the active instance of EVATAR will assume that the connection is either too slow or broken. It will then react to this situation by failing over to the redundant monitoring instance of EVATAR that is running on a different machine using different network connections. This way we achieve better performance and resilience in our system in the case that a connection to EVATAR becomes very slow or completely unresponsive.

Modern households typically contain more than one devices that may host EVATAR such as laptops, desktops, tablets, smartphones etc.

3. Example scenarios

In the first example scenario VAs control LEGO robot Avatars and in the second they bind to a connected collection of sensors and actuators in order to create a simple "Smart Home" application focusing on a home security scenario. These scenarios were implemented to demonstrate EVATAR, for feasibility testing and to demonstrate the applicability of the proposed framework on common, low cost heterogeneous hardware and software.

3.1. Example scenario 1: Collaborative lego robots

The purpose of the implementation of the first example scenario was to test and demonstrate whether the proposed framework is capable of enabling VAs to connect to and control Avatars in the physical world. For the experiment we required a MAS platform that is rich in computational strength, communication and information resources. Also, we required

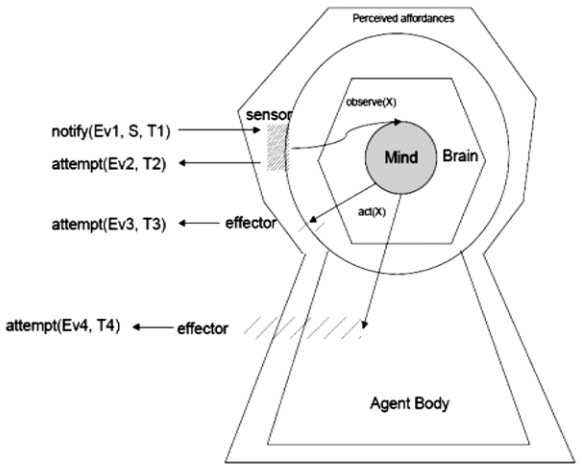

Fig. 7. GOLEM Virtual Agent.

commercial, low-cost robots with limited processing capabilities and resources. We used BIL to describe 2 LEGO robot Avatars [20], 3 VAs in the GOLEM MAS platform [8] and EVATAR to link their body parts and behaviors. Through the experiment we aimed to demonstrate the ability of EVATAR to enable simple and low in computational strength and communication resources robots to present behaviors that would otherwise be impossible to see.

GOLEM is a MAS environment for situating cognitive agents. The agent interaction is mediated by the environment, which is an evolving composite structure supporting Agent–Agent and Agent–Object interaction. The agents in GOLEM (Fig. 7) feature two main components (a) a “mind” component giving the agent reasoning and decision-making capabilities and (b) a “virtual body” containing virtual sensor and actuator components for interaction with objects in the virtual environment. The LEGO Robots included in their body three standard LEGO actuators, a light sensor and a touch sensor. The LEGO robots were running the JAVA-compatible remote control API (LEJOS) [20] that features remote control of the robotic motors and sensors as well as a number of preprogrammed behaviors. The LEGO API allowed us to experiment with both linking virtual sensors and actuators to physical ones as well as with connecting virtual high-level behaviors to LEGO API services.

We programmed the VAs to follow “Observe-Choose-Act” [12] cycles. In Fig. 8 we see the two cycles that describe the “sunny day” scenario in which the VA verifies the connection to the Avatar robotic body that was selected for it and bound to it by EVATAR. Specifically, the VA observes EVATAR for a binding, and as soon as EVATAR informs it that it is bound to a body, the VA selects to test the binding and performs the action (“act”) of testing. In Fig. 8 we see a successful testing scenario that enables the agent to set its state to “embodiment to a robotic avatar body” and store this state in order to start using its Avatar and carry on with its’ duties. The VA tests an actuator (or behavior component relationship) by sending a request to the robot and receiving feedback from the robot regarding the execution of the requested action. The process is similar with sensors but with the difference that the feedback in this case is the appropriate sensory input.

To enable the VAs to perceive the physical world, we rely on anchoring inspired by Coradeschi and Saffiotti [21], viz. a process of establishing a link between sensory perceptual input to an abstract name and maintaining this link for the duration of a task.

When a VA senses a physical object via the sensor of its robotic Avatar, it creates a link between the perceptual input deriving from the object to an abstract name that it assigns to the particular object. This link is stored as an anchor (with perceptual descriptive values e.g. a color) with regards to this object and it may be used and reused in order to recognize this object, communicate about it or perform an action upon it. As we can see in Fig. 9 when an agent is perceiving an object the anchoring process is implemented by replacing the virtual input of the virtual sensor with the input from the robotic sensor that is bound to it through EVATAR and by maintaining the anchoring relationship through the continuous BIL signaling between them. Thus, EVATAR allows the VAs to manifest their behaviors while perceiving objects in a dynamic physical environment through their robotic Avatars. An action in the physical world is achieved when a virtual actuator consumes the corresponding service offered by the robot followed by feedback.

EVATAR is designed for applications that follow the evolution in the state of the agents in the MAS. The GOLEM-EVATAR-LEGO system was situated in an environment with 3 unrecognized objects (balls of different color). The main objective of the VA-Avatar entities was to internalize all the objects in their environment into a simple semantic ontology that would form a lexicon. This lexicon would be used to enable them to communicate with each other regarding the objects in their environment and to allow the third VA without a LEGO Avatar to issue commands to the other two VAs. We call this VA the coordinator.

The coordinator would give the other agents instructions on what to do next. In the beginning of the scenario the coordinator would instruct the VAs to explore the environment for new objects. The agents that were controlling the

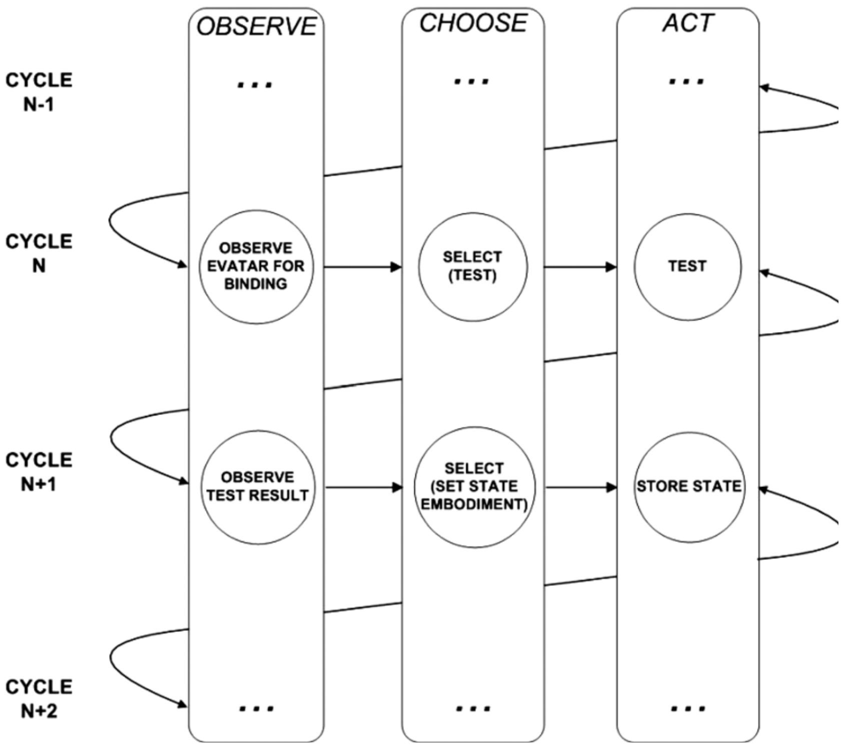

Fig. 8. "Observe-Choose-Act" approach.

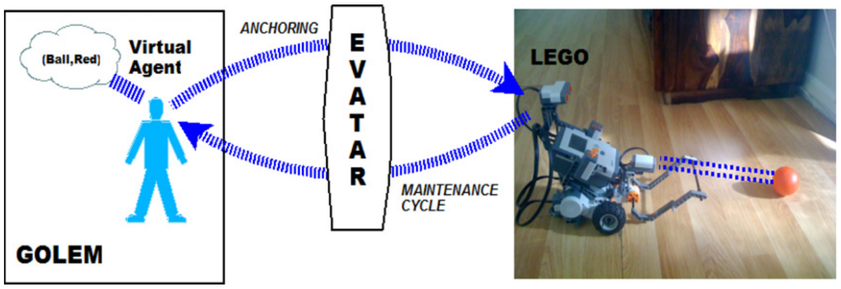

Fig. 9. Anchoring through EVATAR.

robotic Avatars internalized the physical objects in the system by storing their perceptual descriptions (color approximation derived by the light sensor). In the first stage they would send these descriptions to the coordinator. The coordinator would compare approximations sent from all agents and it would construct an ontological lexicon. At a certain point it would instruct the agents to update their local lexicons with the newly constructed one and it would start issuing commands e.g. telling a specific VA to pick a particular object and place it in a particular designated location. The object internalization and perception was feasible due to our adaptation of anchoring.

This was a small experiment but it demonstrated how our framework can enable low-cost commercial robots with limited resources to present more complicated reasoning, planning and communication capabilities. The agent-LEGO robot entities were able to explore their environment, create an ontology and communicate about previously unknown to the system objects.

3.2. *Example scenario 2: "Easy" smart home*

The main idea of this example scenario was to investigate the applicability of our approach to add MAS AI to everyday life by connecting GOLEM VAs to a set of low-cost sensors, actuators, devices and software that are interconnected through a wireless network and distributed in a home environment. We enabled a VA residing in GOLEM to connect to such a set.

There are several cheap and popular wireless technologies that include simple sensors and actuators in order to make a house "smart" and "automated". But the truth is that the cheap hardware allows applications that present limited

Table 1
"Easy Smart Home" hardware.

Hardware	Function	Connection
Generic wireless camera	Motion detection	WiFi
Generic wireless camera	Photo acquisition	WiFi
Waveman wireless receiver	Light switch	Tellstic RF [22]
Waveman wireless receiver	Socket switch (lamp)	Tellstic RF [22]
Generic wireless microphone	Voice recognition	Bluetooth
PC \ laptop	Email\ SMS notification	Wifi\ Internet
PC\ laptop	Time sensor	Wifi\ Internet
Wireless speaker	Sound\ voice	RF

```
select(Home, raise_alarm(Id), T) ←
    instance_of(Activity1, motion, T),
    holds_at(Home, detected, Activity1 , T),
    instance_of(Activity2, authorisation, T),
    holds_at(Home, detected, Activity2 , T),
    holds_at(Activity2, status, failed, T),
    holds_at(Home, alarm, Id, T),
    holds_at(Id, sound, loud, T).
```

Fig. 10. Goal triggered by complex activity.

intelligence. In other words, the "Smart" homes that are commonly created with these cheap hardware are not that "smart". These technologies usually feature connected sensors/actuators using RF (Radio Frequency) communications. In our small experiments for this example case, we used a variety of wireless actuators, sensors and devices to implement a low-cost, "Smart Home" application focusing on security scenarios. Below is a table listing the hardware that was used by our Ubiquitous Agent:

All hardware of Table 1 are either low-cost and easy to acquire or they are abundant and quite commonly found in modern households (laptops, WIFI etc.). The list seems quite small but we can do surprisingly many things if we integrate the contents of Table 1 with GOLEM VAs via EVATAR. So the main idea was to take the above simple hardware components and connect them to Virtual Agents in order to make them collectively present "Smart Home" behavior. They will be the real world Avatar body of the GOLEM VA. We need to note here that we use a "Time Sensor" because the GOLEM VA interprets time based on the agent execution cycles and this should not be confused with the consideration of time in the physical world.

Based on the above setting, we started experimenting on an activity recognition framework. The creation of this framework constitutes an ongoing study and research endeavor of ours and for the purposes of this experiment and example scenario we developed a very basic form of it in order to be able to exemplify the potential of EVATAR for such application. For the implementation of our smart home scenarios we followed a logic-based approach for activity recognition in order to enable the VAs to observe actions based on environmental conditions. In our approach we keep track of all logically consistent explanations of the observed actions. This is done in GOLEM and specifically by defining this in the logic-based reasoning capabilities of the VAs [8] (Fig. 10). We consider all possible plans while all actions and plans are referred to as goals. In general in our approach we borrow elements from the work of Kautz in [23] and from Artikis et al. in [24,25].

All actions in the physical world that are perceived by the agent with an Avatar entity are considered as activities. In our implementation the VA-Avatar coupling perceives two types of activities: simple and complex ones. A perceived simple activity describes the perception of a simple event in an environment e.g. sensing of raw sensory data to detect a change in the luminosity in a room. Complex perceived activities are combinations of perceived simple activities. E.g. The simple activity "Motion Detection" combined with the simple activity "Wrong Password Input" would result to the recognition of the more complex activity "There is an Intruder in the house". The recognition of the more complex activity would instigate an action (from the "library" of possible actions that the agent can select).

We programmed an action policy. These policies are sets of actions that are triggered by simple or complex activities. From the above logic-based rule we can see that the agent has perceived two simple activities, which combined describe a complex activity which is our case is an unauthorized intrusion detection and the agent will execute the appropriate reaction ("start alarm"). By applying an action policy in a GOLEM VA, we are capable of implementing a variety of scenarios.

For the first scenario we applied a "holiday" policy to the GOLEM VA. According to the holiday scenario, the user is going on holiday and the VA takes over by making the house look habited to scare away thieves by turning the lights on and off (wireless light switch) regularly e.g. from 6 pm to 10 pm in the evening (continuing the users normal schedule when he returns from work at 6 pm). If there is an intrusion, the VA will detect it via the augmented motion sensor, take photos using the wireless camera actuator and send an email to the user using the email actuator. The email will be themed with the word "INTRUSION" and it will have attached the photos of the intruder. It will also activate an alarm and the wireless speaker actuator will play a pre-recorded message to the intruder. Similarly, we can have "away to work" policies or "Sleep" policies for protection when the user is asleep during the night.

Another scenario that we tried featured Smart Home automatic light control. According to this scenario, every VA controls the sensors and actuators in a particular room. As soon as the user enters the house, the VA detects him with the motion detection sensor and turns the wireless light switches on. The same will happen if the user leaves one room to enter another. In this case the VA of the second room will inform the VA of the first to turn the light switch off. The first agent will check if there is motion and if not it will proceed with switching the lights off. At all stages the VAs are listening to commands by the user using the wireless microphone sensor. This way they can adjust the light policy according to the users' needs (e.g. the user enters the room but he does not want the light to be on, or he lies on the sofa without motion and he does not want the VA to turn the lights off). Finally, the last parameter of this scenario was the sensing of the physical world time using the "time sensor". The above policies would only take effect when it is dark (based on time of day and expected sunset times).

In general with a limited and heterogeneous set of hardware we proved that we can implement many useful scenarios following our approach with EVATAR, ranging from home security and smart home applications to child monitoring (we could create a policy that notifies the parents what time their teenage son came back home in the night or create a baby monitoring policy).

4. Discussion and related work

In our example scenarios we saw that EVATAR enables VAs to connect to the sensors and actuators of mobile robots or to static, distributed sets of sensors, actuators and smart devices. An interesting point that our implementations revealed was the fact that the level of granularity of the communication between VAs and Avatars affects the overall performance of the system. Therefore it should be considered when designing such architectures. As we have seen in 2.2, EVATAR is designed to support both fine and coarse grain communication. In applications that feature fine grain communication, the events capture low-level actions and communication is essentially a stream of events. Alternatively, in a coarse grain communication application, events represent high-level actions like instructing the robot to "walk 10 m ". There is a relationship between granularity of communication and network speed/bandwidth; a lower speed would be more suitable for coarse grain communications. The same concept applies for limited bandwidth in Avatar hardware.

The performance considerations also led us to the development of BIL instead of using an adaptation of WSDL [15]. WSDL is designed specifically for web services and we do not need all the irrelevant to our implementation information in a WSDL description. BIL uses a more minimalistic syntax framework to decrease BIL message processing overhead, reduce network traffic, increase speeds and in general attain better communication performance.

Furthermore, with regards to the resilience in our proposed distributed architectures, we have presented the "High Availability Clustering" approach that is supported by EVATAR. This approach deals with EVATAR and connectivity failures. In general though, we can identify three categories of failure: agent failure, Avatar failure and the connection failures that we discussed. The first two cases deal with the hardware and software of the agents and their Avatars and they are related to the applications that are deployed by the EVATAR framework. Therefore, the resilience with regards to these types of failure should be considered when developing the VA or Avatar software and it is outside the scope of this paper. Instead, in this paper we focus on EVATAR and how it implements the connections between the agents and their Avatars.

The redundant nodes in our "High Availability Cluster" approach may also run on the same computer as the EVATAR instance. They will still provide failover and resilience if for some reason the active instance of EVATAR fails. The big difference is seen when the failure is caused by external, unpredictable conditions such as loss of TCP connectivity, slow TCP connectivity, computer shutdown etc. If there is an abundance of computers in the network, a redundant monitoring node installed in a different computer to the one of the active instance of EVATAR will be able to take over and restore connectivity between agents and nodes. If there are no failover computers (or smartphones, tablets etc.), and the computer that runs the active instance of EVATAR fails due to e.g. a power shutdown, it would require human intervention to restore the connectivity. When cloud computing becomes more available to the public and specifically for domestic terminals, the above distinction between installations on the same or on different computers will become obsolete.

When discussing the applicability of EVATAR to mobile robots, our case study provided useful feedback regarding the potential of migrating software MAS to augmented reality systems using physical robots. In the introduction we identified middleware frameworks capable of such applications. MIRO the "Middleware for Robots" [6] enables communication among a robot's different modules, between several robots and between a robot and a distributed device such as a PC. Other frameworks are the Microsoft Robotics Studio [5] and the open source Player Stage Project [7], robotic development platforms featuring the simulation of multi-robot behaviors in virtual environments and the ability to reproduce the mechanical behaviors of the robots.

The MARIE [3] and PRISMA [4] approaches produce robotic middleware utilizing a higher level of abstraction. They feature frameworks for integrating new and existing software for rapid prototyping robotic applications in distributed environments. Consequently, they tend to exploit the diversity of communication protocols and mechanisms thus allowing interoperability between different software. EVATAR follows a similar approach as it essentially links processes in MAS and heterogeneous hardware and software. The obvious question at this point would be how does EVATAR compare to other widely used and rich middleware applications such as the aforementioned [3–7].

EVATAR does not intend to compete with these middleware for providing better communication and interfacing between more heterogeneous devices, software platforms, sensors and actuators. Instead it specializes in augmenting existing MAS

VAs for the creation of Ubiquitous Agents and thus incorporating all the benefits of utilizing agents capable of manifesting their behaviors in virtual and physical environments alike. Also, EVATAR allows the creation of affordable applications of this specific type with minimal effort due to its specialization, while the above frameworks would require the development of custom code for such systems. They may enable for example a MAS to send a message to a robot but making this message usable and in general creating an application featuring the integration of the Robots and the MAS into a system would still require both development time and effort.

The ''Middle Layer'' for incorporations among ubiquitous robots by Kim et al. [26] features a conceptually similar approach to EVATAR. The middle layer consists of a sensor and a behavior mapper. The first helps virtual (simulated) robots obtain physical sensor information from mobile robots while the second allows virtual robots to present physical behavior. It is similar to EVATAR since it connects physical and virtual (simulated) robot behaviors. EVATAR provides a framework focusing on exploiting the benefits of integrating VAs into AmI ecologies enabling them to use a variety of sets of physical sensors, actuators and devices as an Avatar (e.g. a smart home or a mobile robot) while dealing with connection issues, system architecture and resilience. On the other hand, the Middle Layer is focusing on abstract connections between robots and their simulations in a virtual environment.

With regards to connecting to sensors, actuators and devices that are distributed within a limited physical environment, EVATAR enables intelligent agents to integrate various technologies creating intelligent applications (e.g. a smart home) using cheap and unintelligent hardware. Wireless low-cost home ''automation'' technologies cannot present this intelligence as the automation and processing resources of their sensors and actuators is usually very limited. Also, since EVATAR provides mediation in a higher level of abstraction, it allows the development of applications that are capable of integrating all of these technologies together into a single system (as we proved in our second example scenario). These agent-Avatar systems present a great potential for integration to larger scale AmI environment/ecologies either via EVATAR (as a SOA middleware) or by using the communication and connectivity resources available to the virtual MAS.

The PEIS ecology [27,2] provides a framework for incorporating smart devices into AmI systems. It utilizes the PEIS middleware by which devices of different types and capabilities (including robots) can cooperate using a distributed tuple-space. The approach focuses on simplifying the communication between the devices while enriching the sensing capabilities in a tagged environment. EVATAR features a complementary approach, as it presents integration potential to such a system in order to provide intelligent agent behavior.

5. Conclusions – future work

In this paper we presented EVATAR and a framework for applying Ubiquitous Agents on robots and smart homes. A Ubiquitous Agent is constructed from a Virtual Agent that provides intelligence with connectivity in a networked environment and Avatars that allow the agent to access and affect the physical environment. The binding between the Virtual Agent and the Avatar has been achieved using the EVATAR middleware, which allows connectivity that extends the functionality of devices with symbolic reasoning, action selection and pro-activity, while at the same time the approach is resilient to faults of the components. The successful prototyping of two demonstration scenarios is both encouraging and has opened up new issues for consideration.

We aim to experiment with EVATAR and obtain further results and metrics regarding the effectiveness of using Ubiquitous Agents in comparison to other approaches and augment more complex MAS agents with a variety of different types of robots and intelligent environments. We will also focus in investigating the potential of applying the concept of the Ubiquitous Agent in a variety of systems and scenarios that could benefit from the ubiquitous presence of an intelligent agent. Also, we will use this framework to study further the requirements and problems that are encountered when enabling entities living in a virtual world and abiding to the laws of this world to sense, act and perceive in a different world (physical). Specifically, we intend to do so through extending the framework of the second example scenario.

References

[1] Harinder Seera, Seng W. Loke, Torab Torabi, Towards device-blending: model and challenges, AINA Workshops, vol. 2, 2007, pp. 139–146.
[2] M. Broxvall, M. Gritti, A. Saffiotti, B.S. Seo, Y.J. Cho, PEIS ecology: integrating robots into smart environments, in: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA, Orlando, USA, 2006.
[3] C. Côté, D. Létourneau, C. Raïevsky, Y. Brosseau, F. Michaud, Robotic software integration using MARIE, International Journal of Advanced Robotic Systems 3 (1) (2006) 55–60.
[4] Jennifer Pérez, Isidro Ramos, Javier Jaén Martínez, Patricio Letelier, Elena Navarro, PRISMA: towards quality, aspect oriented and dynamic software architectures, in: 3rd International Conference on Quality Software, QSIC 2003, IEEE Computer Society, Dallas, TX, USA, 2003, pp. 59–66.
[5] J. Jackson, Microsoft robotics studio: a technical introduction, IEEE Robotics & Automation Magazine 14 (2007).
[6] H. Utz, S. Sablatng, S. Enderle, G. Kraetzschmar, Miro—middleware for placemobile robot applications, IEEE Transactions on Robotics and Automation 18 (4) (2002) 493–497.
[7] B.P. Gerkey, R.T. Vaughan, A. Howard, The player/stage project: tools for multi-robot and distributed sensor systems, in: Proceedings of the International Conference on Advanced Robotics, ICAR 2003, CityCoimbra, Portugal, 2003, pp. 317–323.
[8] Stefano Bromuri, Kostas Stathis, Situating cognitive agents in GOLEM, in: The Proceedings EEMMAS 2007, Dresden, Germany, 2007.
[9] A. Ricci, M. Viroli, A. Omicini, CArtAgO: a framework for prototyping artifact-based environments in MAS, in: Environments for MultiAgent Systems III, in: LNAI, vol. 4389, Springer, 2007, pp. 67–86.
[10] F.L. Bellifemine, G. Caire, D. Greenwood, Developing Multi-Agent Systems with JADE, ISBN: 978-0-470-05747-6, 2007, Hardcover p. 300.
[11] N. Dipsis, K. Stathis, EVATAR-a prototyping middleware embodying virtual agents to autonomous robots, in: Proceedings of the International Symposium on Ambient Intelligence, ISAmI'10, in: Advances in Intelligent and Soft Computing, Springer Verlag, 2010.

[12] Kostas Stathis, A game-based architecture for developing interactive components in computational logic, in: Logical Formalisms for Program Composition, Journal of Functional and Logic Programming 2000 (5) (2000) (special issue).
[13] Thomas Erl, SOA principles of service design, in: The Prentice Hall Service-Oriented Computing Series from Thomas Erl, Prentice Hall PTR, Upper Saddle River, NJ, 2007.
[14] http://en.wikipedia.org/wiki/Wii_Remote.
[15] International business machines corporation, Microsoft. Web Services Description Language (WSDL), http://www.w3.org/TR/wsdl, Ariba, Copyright 2001.
[16] Martin Gudgin Microsoft, Marc Hadley Sun Microsystems, Noah Mendelsohn IBM, Jean-Jacques Moreau Canon, Henrik Frystyk Nielsen Microsoft, Anish Karmarkar Oracle, Yves Lafon, 2007, W3C. SOAP Version 1.2 Part 1: Messaging Framework, second ed. http://www.w3.org/TR/soap/.
[17] Thomas Erl, SOA design patterns, in: The Prentice Hall Service-Oriented Computing Series from Thomas Erl, Prentice Hall/PearsonPTR, ISBN: 0136135161, 2009.
[18] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.
[19] Ev. Marcus, H. Stern, Blueprints for High Availability: Designing Resilient Distributed Systems, John Wiley & Sons, ISBN 0-471-35601-8, 2000.
[20] LEGO co, Mindstorms Robotics Manual, http://www.legomindstorms.com.
[21] S. Coradeschi, A. Saffiotti, An introduction to the anchoring problem, in: Perceptual Anchoring, Robotics and Autonomous Systems 43 (2–3) (2003) 85–96 (special issue).
[22] http://www.telldus.se/.
[23] Henry Kautz, A formal theory of plan recognition, PhD Thesis, Dept. of Computer Science, University of Rochester (Tech. report 215.), 1987.
[24] Alexander Artikis, Marek Sergot, Georgios Paliouras, A logic programming approach to activity recognition, in: Proceedings of ACM International Workshop on Events in Multimedia, 2010.
[25] Alexander Artikis, Anastasios Skarlatidis, Georgios Paliouras, Behaviour recognition from video content: a logic programming approach, International Journal on Artificial Intelligence Tools-IJAIT 19 (2) (2010) 193–209.
[26] T. Kim, S. Choi, J. Lim, Incorporation of a software robot and a placemobile robot using a middle layer, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 37 (6) (2007) 1342–1348.
[27] A. Saffiotti, M. Broxvall, PEIS ecologies: ambient intelligence meets autonomous robotics, in: Proc. of the Int. Conf. on Smart Objects and Ambient Intelligence, sOc-EUSAI, Grenoble, France, 2005, pp. 275–280.