Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Computer Communications xxx (2006) xxx–xxx
computer communications
www.elsevier.com/locate/comcom

Agent based middleware infrastructure for autonomous context-aware ubiquitous computing services

John Soldatos [a,*], Ippokratis Pandis [a], Kostas Stamatis [a], Lazaros Polymenakos [a], James L. Crowley [b]

[a] *Athens Information Technology, 19,5 km Markopoulo Ave., P.O. Box 68, GR-19002 Peania, Greece*
[b] *INRIA Rhone-Alpes, 655 Ave de l'Europe, 38330 Montbonnot-St. Martin, France*

Abstract

Middleware for ubiquitous and context-aware computing entails several challenges, including the need to balance between transparency and context-awareness and the requirement for a certain degree of autonomy. In this paper we outline most of these challenges, and highlight techniques for successfully confronting them. Accordingly, we present the design and implementation of a middleware infrastructure for ubiquitous computing services, which facilitates development of ubiquitous services, allowing the service developer to focus on the service logic rather than the middleware implementation. In particular, this infrastructure provides mechanisms for controlling sensors and actuators, dynamically registering and invoking resources and infrastructure elements, as well as modeling of composite contextual information. A core characteristic of this infrastructure is that it can exploit numerous perceptual components for context acquisition. The introduced middleware architecture has been implemented as a distributed multi-agent system. The various agents have been augmented with fault tolerance capabilities. This middleware infrastructure has been exploited in implementing a non-obtrusive ubiquitous computing service. The latter service resembles an intelligent non-intrusive human assistant for conferences, meetings and presentations and is illustrated as a manifestation of the benefits of the introduced infrastructure.

Keywords: Middleware; Autonomic computing; Pervasive computing; Ubiquitous computing; Context-awareness; Smart spaces

1. Introduction

Middleware deals with the coordination, cooperation and interoperability of distributed components through bridging the gap between applications and their underlying low-level software and hardware infrastructure. Moreover, it facilitates integration of components in distributed heterogeneous environments. Middleware systems and architectures are becoming increasingly important as networks, services and applications become more complex. These architectures provide a basis for tackling stringent requirements regarding faster development and cost-effective operation. The latter requirements expand the scope of middleware to address not only faster development, deployment and integration, but also cost-effective systems operation and management. To this end, emphasis is put on designing, developing and deploying active systems, which feature autonomic existence and are commonly classified as autonomic. Autonomic computing systems possess several characteristics including that they are self-defining, self-configuring, self-optimizing, self-healing, context-aware and anticipatory. Middleware services and architectures are gradually evolving to support autonomic computing systems [1].

Several research issues come into foreground, when it comes to supporting the visionary, yet constantly evolving ubiquitous computing paradigm [2]. Ubiquitous computing services aim at exploiting the full range of sensors and networks available to transparently providing services, regard-

* Corresponding author.
E-mail addresses: jsol@ait.edu.gr (J. Soldatos), ipan@ait.edu.gr (I. Pandis), ksta@ait.edu.gr (K. Stamatis), lcp@ait.edu.gr (L. Polymenakos), James.Crowley@inrialpes.fr (J.L. Crowley).

0140-3664/$ - see front matter © 2006 Published by Elsevier B.V.
doi:10.1016/j.comcom.2005.11.018

less of time and end user's location [3,4]. A core characteristic of pervasive and ubiquitous computing systems is that they are context-aware, in the sense that they are able to provide services not only based on information that end users provide, but also based on implicit contextual information [5]. Implicit information is usually derived based on a rich collection of casually accessible, often invisible sensors that are connected to a network structure. Apart from context-awareness, ubiquitous computing systems feature increased dynamism and heterogeneity, which differentiate them radically from traditional distributed systems. The underlying ubiquitous computing infrastructures are more sophisticated and bring into foreground issues such as user mobility, disconnection, dynamic introduction and removal of devices, diverse network connections, as well as the need to blend the physical environment with the computing infrastructure [6]. Ubiquitous computing components are related to autonomic computing, since autonomy is a key to confronting these challenges. All major pervasive and ubiquitous computing projects (e.g. [7–11]) have built sophisticated middleware infrastructures. These projects reveal that middleware for ubiquitous computing is much more complex comparing to conventional distributed systems. However, they are focused on a specific set of middleware services facilitating their target applications. For example, some emphasize on context-awareness, others on transparent communications and mobility, while some others concentrate on autonomy. In this paper we describe a middleware infrastructure addressing a wide range of issues entailed in ubiquitous computing services. Specifically, this infrastructure provides mechanisms for service access, context modeling, control of sensors and actuators, directory services for infrastructure elements and services, as well as fault tolerance. We describe this infrastructure with particular emphasis on a framework for controlling sensors and actuators, as well as our approach for modeling situation states. Also, we describe the implementation of this framework over an agent platform. Overall this middleware infrastructure allows ubiquitous service developers to focus on the service logic of the implementation, rather than implementing the middleware. The various frameworks provide functionality that can be reused across different ubiquitous computing services.

Based on the introduced middleware platform, we have built a prototype ubiquitous computing service, namely the Memory Jog (MJ), which resembles a smart non-intrusive assistant for meetings and conferences. This service is built in the scope of a smart room, which comprises a rich sensing infrastructure comprising multiple sensors. A number of perceptual components such as for face detection and recognition, acoustic localization, person tracking and speech activity detection were implemented over this sensing infrastructure. These perceptual components were accordingly used to support context-awareness based on the introduced context modeling approach. In particular, perceptual components outputs were combined with a view to identifying composite contextual states. Note that perceptual components were wrapped as agents and accordingly integrated to the rest agent based middleware framework.

The service logic of the Memory Jog made use of the introduced sensor and actuator control framework with a view to dynamically discovering hardware and software elements, and invoking their services. This framework facilitated the implementation of the Memory Jog service logic given that important middleware services were reused. Indeed, by reusing middleware services the Memory Jog service developers allocated effort on implementing the service logic, paying special attention in usability aspects, such as the intuitiveness of the user interface and the non-obtrusive nature of the service. These aspects were positively evaluated in the scope of simulation studies with end users. Main conclusion and results from these studies are also included in this paper.

The rest of the paper is structured as follows: Section 2 provides a taxonomy of middleware components for ubiquitous computing. Section 3, introduces our overall middleware architecture for ubiquitous computing services and positions it with respect to other prominent middleware frameworks for ubiquitous computing. Special emphasis is paid into describing our approach for context modeling, as well as a framework for dynamically controlling sensors, actuators and services. It is also illustrated that this middleware infrastructure was implemented as a distributed multi agent system. Section 4 describes presents the implementation of the Memory Jog service based on the introduced infrastructure. It also reports main results from simulation studies involving users. Finally, Section 5 summarizes the paper and outlines the main conclusions.

2. Taxonomy of middleware components for pervasive computing

Middleware architectures for traditional computing services aim at providing complete transparency of the underlying technology and their surrounding environment. While this provides several benefits it is not the ultimate goal in ubiquitous computing environments. These environments target context-awareness, which demands availability of knowledge and information about the surrounding environment. At the same time there is also a need for an appropriate degree of transparency, since this can reduce software complexity and optimize the use of system resources. As a result, ubiquitous computing middleware strives to achieve an optimal balance between awareness and transparency [2].

Other objectives of middleware architectures and components are to ease application developers in exploiting their capabilities. Efficient middleware architectures facilitate structured integration of components based on well-defined development processes and programming environments. Note however, that the efficiency of middleware components is audited based on the quality of their runtime services. As a result, middleware enables the cooperation between development support and runtime services. This cooperation is particularly difficult in the scope of pervasive computing, given that middleware components expose multiple interfaces to different application level components, while also providing a multi-facet runtime support. In particular, components supporting ubiquitous computing can be classified according to their functionality, as illustrated in the following paragraphs.

2.1. Transparent ad hoc communication

Middleware components in ubiquitous computing provide transparent communication between the diverse sensors and devises engaged in the computing infrastructure (e.g., cameras, microphone, computers, PDAs, smart phones). Middleware components abstract the details of communication channels and protocols and achieve interoperability regardless of the underlying network infrastructure. As devices are added and/or removed from the network, systems and applications are notified. Publish-subscribe mechanisms and popular XML messaging protocols can be employed to this end.

2.2. Capture and transfer of sensor streams

Capturing sensor data is a prerequisite to obtaining information about the surrounding environment. To this end, low level middleware components interface with the various sensors in order to obtain raw sensor data. Such components include a rich set of capture drivers for different sensors.

In the scope of ubiquitous computing applications, raw sensor data is processed towards extracting context cues. In most cases this processing is performed at different computing platforms that the host capturing data (Fig. 1). This is mainly due to the need to exploit distributed computational power given that sensor processing might be computationally demanding. Therefore, there is a need for additional components undertaking the graceful transfer of sensor streams across the network for distributed processing. Representative components falling in this category are high performance sockets ensuring quality of service in the delivery of sensor data. A prominent example of such a middleware infrastructure is the NIST Smart Flow System [12,13].

2.3. Raw signals processing

Raw sensor data is processed and contextual information relating to location, identity and activity is obtained. Such information constitutes a form of *elementary* context, but it is important since it can serve as an anchor to deriving additional information [3]. Collecting elementary context hinges on middleware components performing computationally complex signal processing on the sensor data (e.g., audio, visual streams). Such middleware components include a wide range of perceptual technologies (e.g., person and object identification, people and object tracking, multimodal interactions, speech recognition, body tracking).

2.4. Context acquisition – situation recognition

Context-awareness in ubiquitous computing is not limited to identifying people, objects and their locations. On the contrary, the emphasis is on identifying situations composed of multiple forms of elementary context. As a result, middleware components for modelling and dynamically detecting situations are important to any non-trivial ubiquitous computing service. Conventional programming languages provide limited or no support for context-awareness. Furthermore, technologies providing support for context-awareness are likely to present differences across different programming languages. This creates portability problems for context-sensitive applications, which middleware architectures attempt to solve. Thus, middleware must provide a uniform and common way to express

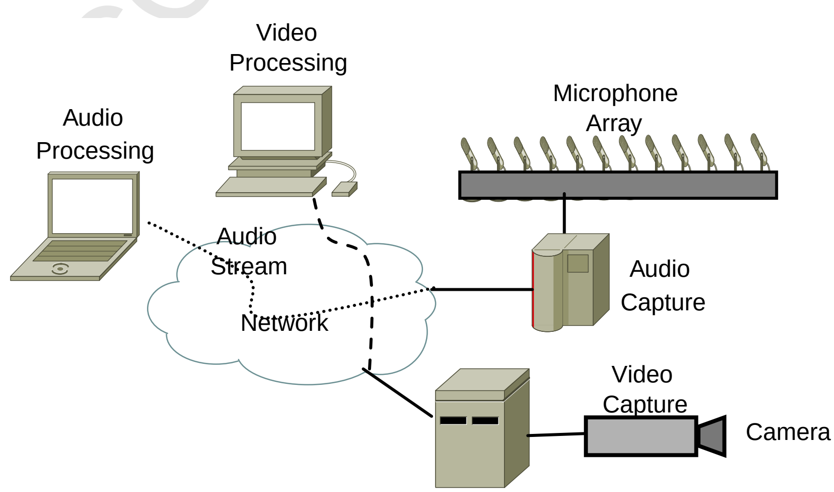

Fig. 1. Capture, transfer and distributed processing of sensor streams.

the software's context-awareness with minimal dependencies on specific languages, operating systems, sensors or environment.

2.5. Decision making – context triggered service logic

Context acquisition and situation recognition constitute prerequisite steps in implementing the service logic. Service logic in traditional applications is triggered on-demand paradigm, i.e., upon users' requests. This paradigm is essentially augmented in the scope of ubiquitous computing applications, since the service logic can also be triggered automatically, based on the current context. Automatic triggering may involve adapting to the new environment, notifying the user, as well as communicating with other computers or devices to exchange information. Context-triggered service logic is a foundation for non-intrusive services.

3. Middleware infrastructure for ubiquitous computing

3.1. Related work

This section presents key elements of a middleware infrastructure devised and developed in the Computers in the Human Interaction Loop (CHIL) project [14], with a view to easing service development and application integration. CHIL emphasizes on the development of ubiquitous, context-aware services in in-door environments, which are equipped with numerous sensors (i.e., microphones and cameras). These environments are conveniently called '*smart rooms*'. Fig. 2 depicts the floor plan of one of the four smart rooms that have been setup in the CHIL project, namely the Athens Information Technology smart room. Services developed in these smart rooms comprise a large number of perceptual middleware components (such as recognition and localization algorithms), which provide contextual information on people and objects' identity and location. Specifically, CHIL service developers exploit a wide range of perceptive interface components including a rich collection of 2D-visual components (i.e., person localization and tracking, body detection, head orientation, face detection and recognition), 3D-visual perceptual components (i.e., person tracking, gesture/posture recognition, head & hand tracking using stereo cameras, pointing gesture recognition using stereo cameras), acoustic components (i.e., speech recognition (including far-field), source localization, speech detection, speaker identification, acoustic emotion recognition, acoustic event classification, beamforming), as well as audio-visual components (i.e., A/V person tracking, person identity tracking, activity recognition, AVSR – mouth (lips) observation, emotion recognition). The middleware infrastructure presented in this section facilitates integration of these components, as well as the fusion of their contextual information with a view to deriving more sophisticated context. The diversity of these technology components, the potential sophistication and integration complexity of the services, as well as the number of collaborating organizations and demonstration sites, pose unique integration challenges.

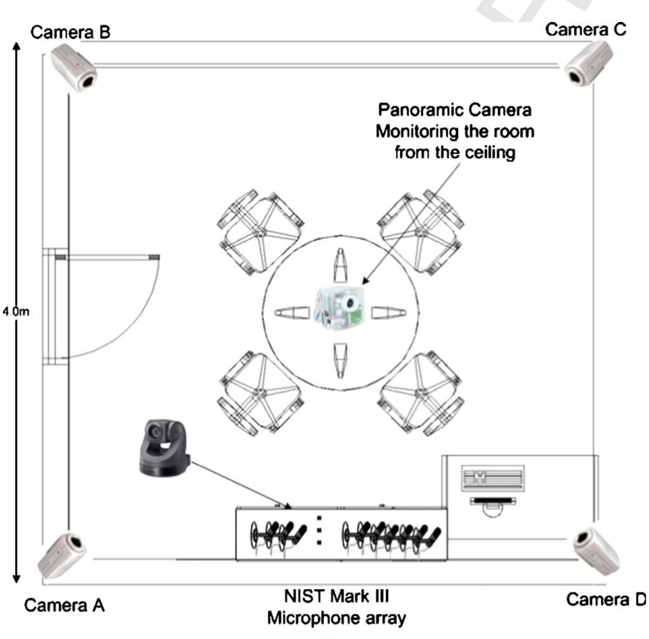

Fig. 2. Floor plan and sensors of the Athens Information Technology Smart Room.

All non-trivial ubiquitous and pervasive computing projects have devised similar middleware infrastructures. The Interactive Workspaces project at Stanford University [7] focused on human interaction with devices and large high-resolution displays. A key challenge in this project is the coordination of multi-modal, multiuser and multi-device applications in different contexts. To this end the project has developed the Interactive Room Operating System (iROS) [15], which provides a reusable, robust and extensible software infrastructure enabling the deployment of component based ubiquitous computing environments. IROS supports various modalities and human-computer interfaces, by tying together devices each one having its own operating system.

The Oxygen project at MIT concentrates on a pool of user and system technologies enabling pervasive human-centered computing. In Oxygen applications special emphasis is paid on automated, personalized access to information, adapting the applications to users' preferences and needs. In terms of middleware architecture, the Oxygen project has produced the MetaGlue system [10], which constitutes a highly robust agent platform, where agents represent both local resources and interactions with those resources. Metaglue relies on a custom distributed communication infrastructure enabling agents to run autonomously from individual applications so they are always available to service multiple applications. Metaglue is efficient in implementing autonomous agents that significantly augment the functionality of the space and facilitate user interaction. However, it provides no essential support for implementing context-awareness. The latter is addressed in the GOALS architecture [9], which is the evolution of the MetaGlue system.

The EasyLiving system developed at Microsoft research is another prototype ubiquitous computing architecture [16]. Easy Living focuses both on the coordination of the devices, but also on exploitation of contextual information. Specifically, the system employs computer vision technologies for person-tracking and visual user interaction and supports context-awareness based on a geometric model of the world. It uses device-independent communication and accordingly adapts the user interface.

The Aura system [8] targets pervasive computing environments involving wireless communication, wearable or handheld computers, and smart spaces. Aura provides software architectural models that monitor an application and guide dynamic changes to it. Thus, it provides opportunities for adapting to varying resources, user mobility, changing user needs and system faults.

The fact that each of the above projects has built its own infrastructure manifests that there is no global unified framework addressing all needs. Architectures tend to concentrate on particular application aspects. Some focus on the co-ordination of physical space and devices (e.g., interactive workspaces), others on synchronizing multiple modalities (e.g., Oxygen), and others on user mobility and attention (e.g., AURA). Nevertheless, there is no architecture providing the necessary level of sophistication for supporting integration of a large number of autonomic perceptual components, which is a major research challenge in CHIL.

3.2. Agent platforms

In order to alleviate the complexity of building middleware for ubiquitous computing, we have strived as much as possible to exploit pre-existing platforms and components. In particular, we have taken advantage of middleware developments supporting high performance transfer and processing of streams, context-awareness and situation detection, transparent ad hoc communication, as well as autonomic features. These components have, however, been appropriately customized towards implementing a dynamic self-resilient infrastructure for provision of services, along with a powerful mechanism for sophisticated context modeling.

At the heart of our middleware infrastructure implementation is a distributed agent infrastructure. Agent infrastructures facilitate the implementation of communication between distributed entities based on rich semantics (see, for example [17,18]). Moreover, they ease the implementation of transparent ad hoc communication between distributed components. Furthermore, agents provide a certain degree of autonomy (e.g. [19]), which constitutes a sound basis for implementing autonomic features.

Software agents lack the capabilities required to support high performance transfer of sensor streams. Infrastructures for distributed transfer of sensor streams are usually built as system level components that do no feature the high level capabilities of software agents. There is therefore a need for integrating low level stream transfer middleware with agent capabilities. A prominent way to achieve this is to wrap low level middleware components with agent based middleware, so that they behave as software agents. The concept is depicted in Fig. 3, which shows that low level components become part of the agent infrastructure, as soon as an agent wrapper is implemented on top of them. As all middleware components expose agent behavior, they can be managed based on a single higher layer interface. Note that in Fig. 3, middleware components can be distinguished into two basic sets according to their socket communication capabilities. Higher performance sockets are required for the distributed transfer of sensor streams, while agents com-

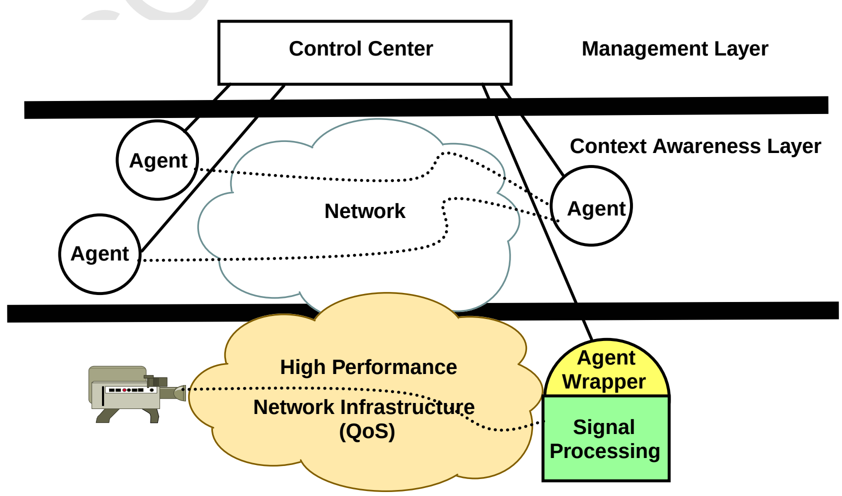

Fig. 3. Combining sensor processing and context-awareness.

municate through conventional socket interfaces. This is illustrated in the figure in the form of two logically distinct network infrastructures, which, however, correspond to the same physical network connectivity.

3.3. Middleware system overview

Fig. 4 depicts an anatomy of a multi-agent framework supporting the implementation of ubiquitous and pervasive computing services. Specifically, this framework provides a set of functionalities that along with the sensing infrastructure can be re-used across different ubiquitous computing services. These functionalities include mechanisms to:

- Control the sensors and actuators of the 'smart room'.
- Control user access to services.
- Modeling composite contextual states based on combinations of perceptual components.

Several ubiquitous computing services can leverage this reusable functionality, which allows the service developer to concentrate on implementing the service logic rather than the middleware. Apart from this set of reusable components and services, the framework implements 'pluggable' mechanisms for incorporating additional perceptual components and sensors.

The framework consists of the following agents types:

- *Core Agents:* Core agents are independent of the service and smart room installation independent. They provide the communication mechanism for the distributed entities of the system. Moreover, core agents undertake the control of the sensing infrastructure, while also allowing service providers to '*plug*' service logic into the framework. Core agents include the:
  - ∘ Device Desktop Agent, which implements the user interface required for accessing the ubiquitous services. A 'pluggable' mechanism allows the user interface to be customized to the particular ubiquitous computing service.
  - ∘ Device Agent, which enables different devices to communicate with the framework.
  - ∘ Personal Agent, which constitutes the proxy of the end user in the agent world. The personal agent conveys user requests to the agent manager, which are accordingly handled by appropriate agents. It maintains the user's profile in order to personalize the services to the end user.
  - ∘ Agent Manager, which allows the system to be dynamically augmented with additional Service Agents. Thus, the Agent Manager allows additional basic, as well as ubiquitous computing services to be incorporated to the system.
- *Basic Services Agents:* These agents incorporate the service logic of basic services, which are tightly coupled with the installed infrastructure of each smart room. Basic services include the ability to track composite situations, as well as the control of sensors and actuators. Tracking of composite situations is performed through the Situation Watching Agent (-SWA) (Fig. 4) based on the context modeling approach discuss in following paragraphs. Also, control of sensors and actuators is performed through the Smart Room Agent in a way that is also elaborated in subsequent paragraphs. Furthermore, a Knowledge

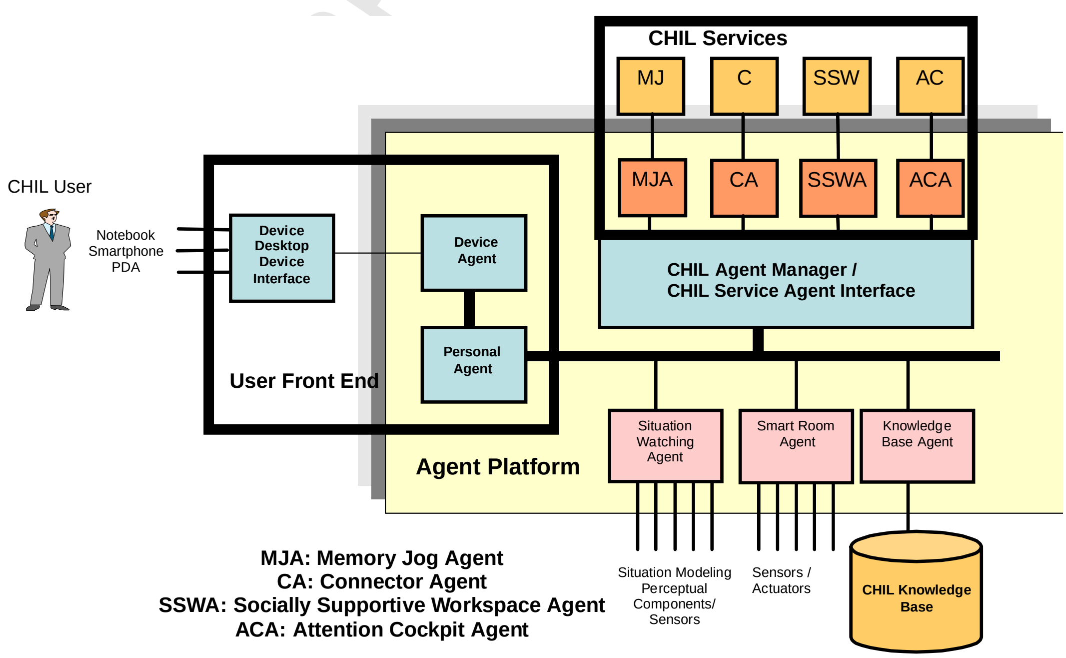

Fig. 4. Middleware infrastructure for ubiquitous computing.

Base Agent, allows the agents of the framework to dynamically access information on the state of the components of ubiquitous computing environment (-e.g., sensors, actuators, perceptual components), through a Knowledge Base Server that is supported as an ontology management system.

- *Ubiquitous Service Agents:* Ubiquitous service agents implement the non-obtrusive service logic of the various context-aware services. Each ubiquitous computing service is therefore implemented as a Ubiquitous service agent and accordingly plugged into the framework. In the scope of the CHIL project, several ubiquitous agents corresponding to various ubiquitous computing services are implemented and integrated into the framework. A following section elaborates on the MJ service, which is implemented through the Memory Jog Agent (MJA). Fig. 4, depicts also the Connector Agent (CA), the Socially Supportive Workspaces Agent (SSWA), and the Attention Cockpit Agent (ACA), which correspond to other CHIL services.

This agent framework has been implemented based on the Java Agent Development Environment (JADE) platform [20]. In this implementation, agent communication is realized based on Foundation for Intelligent Physical Agents (FIPA) primitives [21]. Several aspects of this agent based middleware framework are described in [22]. Moreover, information about the Knowledge Base and its use as a directory service for middleware components and services is provided in [23]. Following paragraphs describe the approaches adopted for context modelling and sensor/actuator control, while also illustrating how agents have been augmented with autonomic capabilities.

3.4. Context modeling

Context modeling middleware facilitates ubiquitous computing services with the ability to describe the state of their surrounding environment, while also providing mechanisms for accessing this description.

Accordingly, context modeling languages exploit this middleware to encode the detection of events that are necessary to initiate or terminate service actions. There are several approaches to modeling situations, which according serve as basis for implementing context-aware components.

The approach adopted and used along with the agent middleware infrastructure of the previous paragraph is based on the notion of networks of situation states [24]. According to this approach a situation is considered as a state description of the environment expressed in terms of entities and their properties. A situation is a kind of state description composed of a conjunction of predicates. Predicates are truth functions that can take on logical or probabilistic values. Situations are defined in terms of an assignment of observed entities to 'roles', the properties of the entities assigned to roles, and the relations (i.e., relative properties) of the entities playing roles.

Entities have numerical attributes such as position, orientation, size, configuration or external appearance. These are tracked by perceptual components and can be used to compute relations. A relation is a predicate (truth) function computed over the attributes of one or more entities. Relations may be represented by boolean or probabilistic truth-values. Each situation is defined in terms of a set of roles and relations. The concept of role is an important tool for simplifying the network of situations. It is common to discover a collection of situations for an output state that have the same configuration of relations, but where the identity of one or more entities is varied. A role serves as a 'variable' for the entities to which the relations are applied, thus allowing an equivalent set of situations to have the same representation. A role is played by an entity that can pass an acceptance test for the role, in which case, it is said that the entity can play or adopt the role for that situation.

A situation model describes activity using a network of situations. Such a model specifies the entities, properties and relations that must be observed towards triggering the service logic. Changes in individual or relative properties of specified entities correspond to events that signal a change in situation. For example, in the scope of a meeting involving short presentations, at any instant, one person plays the 'role' of the 'presenter', while the other persons play the role of 'attendees'. Dynamically assigning a person to the role of 'presenter' makes it possible to select perceptual component to acquire images and sound of the current speaker. Detecting a change in some role allows the system to reconfigure the video and audio acquisition systems.

Situation models determine the entities to observe, the properties to measure and the events to detect, and thus specify the selection and configuration of perceptual components (i.e., components realizing lower level signal processing). Accordingly, perceptual component outputs can be combined to identify situation states of the situation model, as shown in Fig. 5.

An example of a situation model targeting context-awareness for meeting activities involving an agenda and presentation is depicted in Fig. 6. This model signifies the importance of the following events with respect to a meeting:

- Commencement of the meeting.
- Start of the presentation on a particular agenda item (i.e., session of the meeting).
- Questions on each of the presentations.
- End of the presentation.
- End of the meeting.

Moreover, this model defines possible sequences of occurrence for these events, based on the arcs connecting the various situations. The context-aware middleware encoding this situation model makes provisions for both recognizing situation and situation transitions, but also

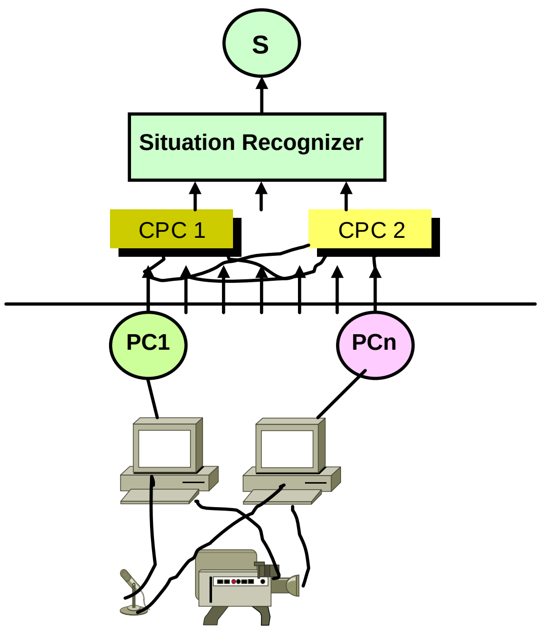

Fig. 5. Situation detection.

for triggering service logic associated with each of these situations.

Describing context as a network of situations may seem limiting and not scalable, mainly because it is unlikely to capture rich context based on a small set of situation states. Nevertheless, a situation model can be dynamically extended as new types of relations between entities are identified. Furthermore, there is always a possibility for making use of more than situation models in the scope of an application. Extending the situation model dynamically, while also dynamically switching between more than one model provides significantly more expressing power.

The network of situations approach has been implemented in the Situation Watching Agent of our framework. In particular, the Situation Watching Agent parses situation models that are expressed in XML format. Each situation model reveals the perceptual components and their configuration required to identify each state of the model. Once a situation model is loaded to the Situation Watching Agent (based on an appropriate XML file), the Situation Watching Agent parses the model and identifies the perceptual components required to track the states of the model. Accordingly, the SWA conveys requests for subscribing to these perceptual components to the Perceptual Components Wrapper Agent (PCWA). The PCWA queries the directory services (i.e., the knowledge base) to dynamically discover the properties and configuration of perceptual components, and then subscribes to them. The required perceptual components provide input to the PCWA, which acts also as a manager of these subscriptions. As the perceptual components send their output to the PCWA, the latter filters these outputs according to the properties of the subscription and forwards them to the SWA. The whole process is illustrated in Fig. 7. Thus, the Situation

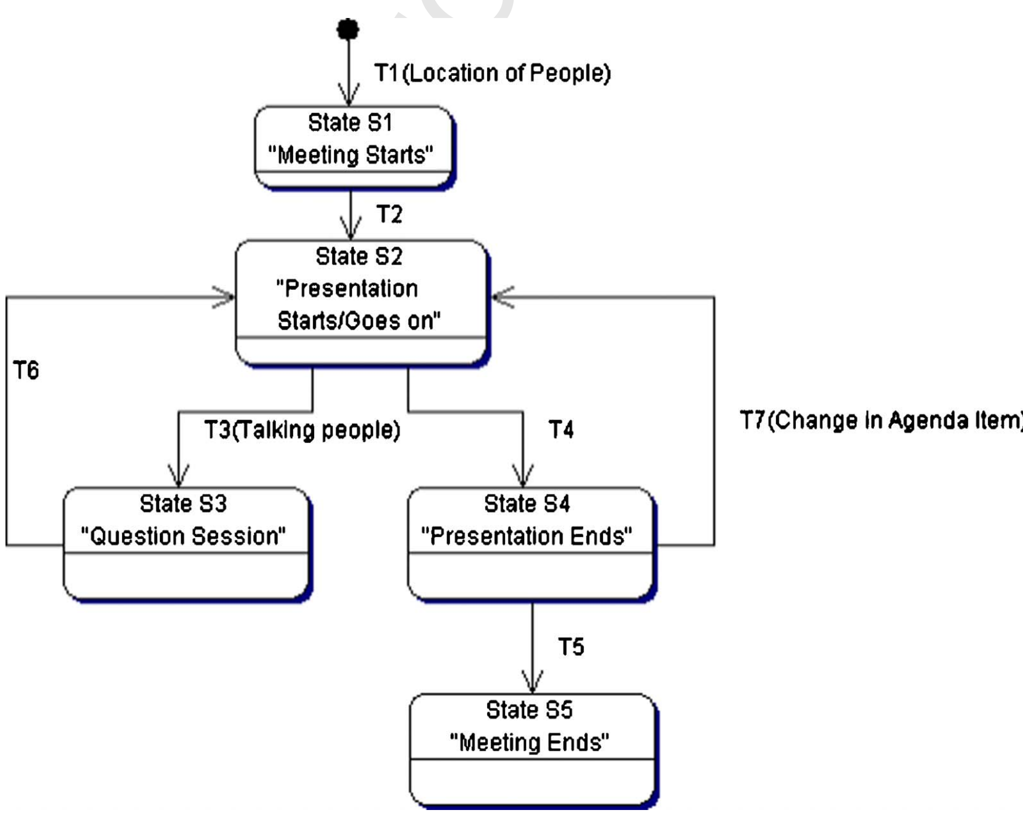

Fig. 6. A sample situation model.

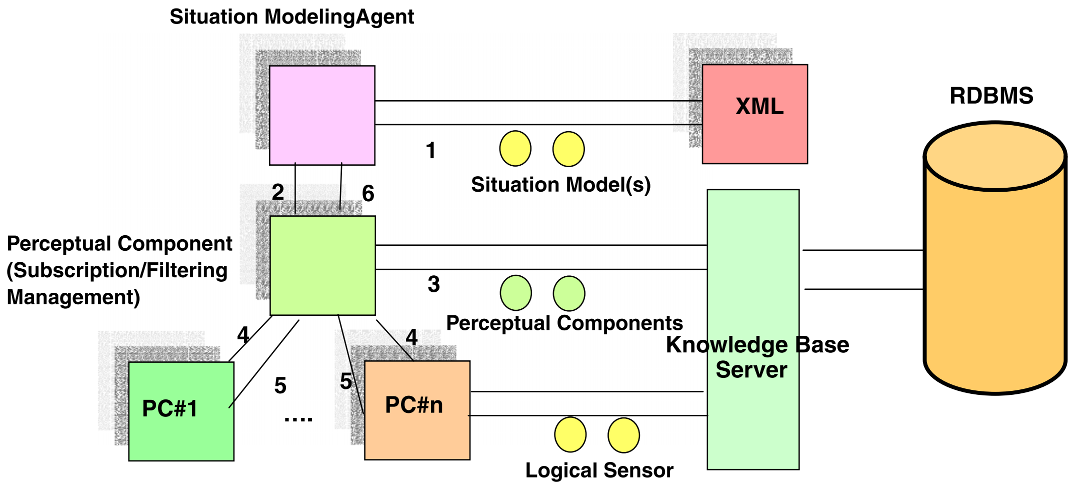

Fig. 7. Combining perceptual components to identify situation model states.

Watching Agent acts as a context broker, which is a quite common approach in context-aware architectures for smart spaces.

3.5. Agent based service oriented infrastructure – sensor and actuator control

The introduced middleware infrastructure provides a common interface (API) for accessing and controlling the various hardware elements (i.e., sensors and actuators). To this end, sensor and actuators register with the directory service provided by the Knowledge Base Service. Sensor and actuator meta-data, which are registered within the knowledge base server, include information about the vendor, the model, the status, interfaces, capabilities, as well as the network addresses of the device. From an implementation perspective, we have concentrated on registering the two main types of sensors that exist in our smart room (Fig. 2), namely microphones and cameras. Thus, we have implemented three distinct proxy agents for these devices: one generic, one for microphones and one for cameras. The main responsibilities of these proxies are to:

- Represent sensors and actuators in the world of agents and provide access to the rest of the framework.
- Interact with the directory service of the knowledge base.

For each new device (i.e., sensor or actuator) that is installed in the room, a new proxy agent is instantiated as a mean to controlling the device. This proxy agent constitutes an agent wrapping to the device control capabilities. Upon the initialization of the device, the proxy agent is responsible for registering it with the knowledge base. Accordingly, it updates the indicated operational state of the device in the registry (for example, when the device shuts down or restarts). Finally, it translates requests from other agents of the framework, to device-specific calls.

Similarly to the infrastructure elements the framework controls various infrastructure specific (auxiliary) services. Developers of ubiquitous computing applications use the framework to dynamically access information on the available value-adding services installed in the infrastructure. Prominent examples of such services include a text-to-speech (TTS) service, a display, and a targeted audio service. Information about these services is registered using a proxy agent, similar to the case of sensor and infrastructure elements registration. The mechanism is illustrated in Fig. 8. A wrapper agent represents the services available to the agent platform, enables communication with the rest of the framework, translates requests from the various clients to service-specific calls and interacts with the knowledge base. This wrapper agent provides another level of abstraction. Specifically, all services that provide the same functionality (e.g., all TTS services) are wrapped by a service proxy of the same type (e.g., a TTS proxy). This service specific proxy handles all requests for that service, being also responsible to forward them to specific implementations and machines that host this service. The service proxy retrieves also dynamically information (from the knowledge base) about the existence, the properties and the operational status of the available services. In the case where there is no available provider of this service and the proxy declares incapable of fulfilling the request.

Note that the particular algorithm for selecting a service implementation depends on the targets and goals of the overall ubiquitous computing service. For example, a TTS service instance, as well as a display service instance may be selecting by the corresponding proxies based on a variety of criteria involving people locations an orientation within the smart room.

Fig. 8 illustrates the implemented registration mechanism enabling discovery and manipulation of services and infrastructure elements. The mechanism involves the following steps:

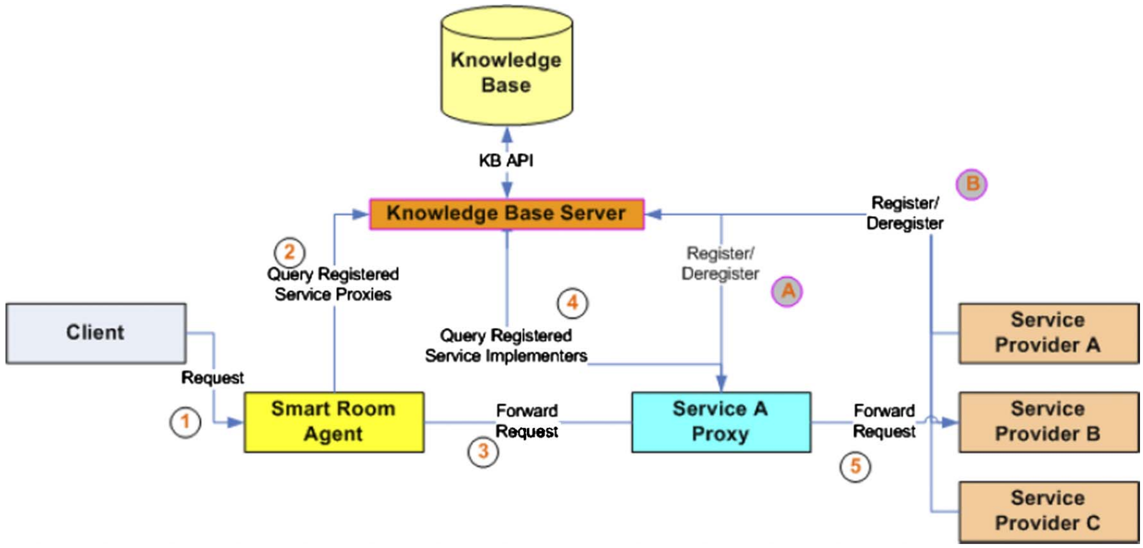

Fig. 8. Registration, dynamic discovery and invocation for the services of the system.

- The proxy of a specific service registers into the system (step A).
- All the providers of this specific service also register themselves into the system (step B).
- When clients want request a particular service invocation, they send a request to the gateway for all the services (step 1), which is a dedicated agent and is called the Smart Room Agent (SRA).
- The SRA searches the registry in order to see if there is a proxy for such a service (step 2).
- Assuming that a proxy is found it forwards the request to it (step 3).
- When the service proxy receives a new request, it checks the registry to find available service providers (step 4).
- A selection algorithm is used to decide to which service provider to forward the request. Following the selection the request finally is received and served by a service provider (step 5).

Note that the all information is dynamically looked up at the knowledge base. This is performed to support for service providers dynamically coming into and going out of the system.

3.6. Autonomic features

Agent platforms support certain autonomic features of a distributed system, including the abilities to persist, clone and move (migrate) components to other hosts. However, there is also a need to implement application specific functionality for discovering agent deficiencies, since the later are differently defined in the scope of an application.

Based on the JADE platform we augmented all agents of the framework with the capability of querying agent components about their status. Thus, we implemented a '*ping*'-like functionality for all agents of the framework. Moreover, as agents discover the status of other agent entities, we have implemented functionality enabling agents to adapt their behavior to the status of other agents. This is particularly important in the case where the availability of an agent entity is a prerequisite for the operation of others. Specifically, in the middleware framework presented in Fig. 4, several agents depend on others. For instance, the Situation Watching Agent relies on underlying wrappers of perceptual components to support situation recognition. In general, an agent has a set of dependencies expressed as a dynamic list of other agents. As a first step to ensuring autonomy and maximum service availability of the system, we implemented functionality allowing every agent to keep track of the list of its dependants and accordingly adapt its functionality. Adaptation results in downgrading or upgrading the functionality and features offered by the particular agent, depending on the availability of other agents.

As a second step to autonomy we provided middleware for self-healing functionality. This was achieved through migrating dependant agents to a different execution environment (e.g., machine or agent container) upon detection of problems with their availability. To this end the migration process is combined with the detection ('*ping*') functionality outlined above. Agent migration is undertaken from another entity that is able to detect the problem. The delegation of this entity is implemented based on either an Autonomic Manager agent entity, which undertakes the role of migrating and restarting agents. The autonomic manager exploits the 'ping' functionality to detect failing agents. Fig. 9 depicts the state diagram of an agent incorporating autonomic functionality. This agent 'pings' dependant agents and accordingly modifies its state.

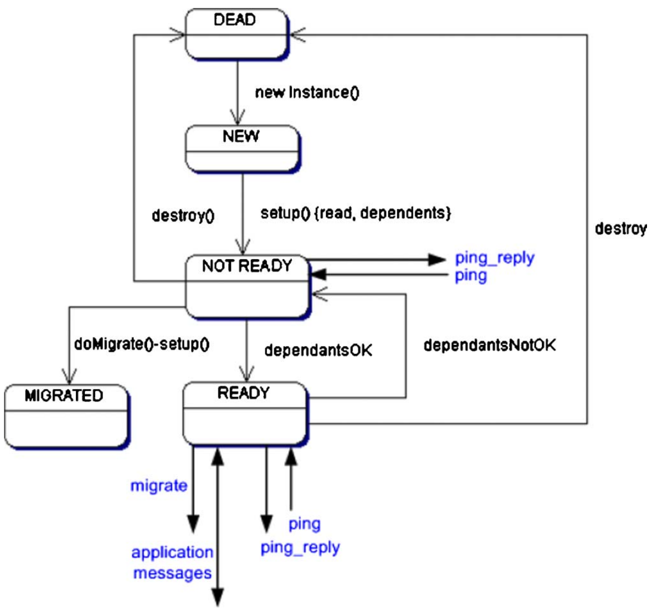

Fig. 9. Pinging dependent agents and agent migration.

4. Ubiquitous computing application implementation

4.1. Overview of the Memory Jog service

The middleware infrastructure outlined in the previous section served as a basis for implementing ubiquitous computing services. In the sequel we present the implementation of an application constituting a non-intrusive assistant for events such as lectures, meetings, presentations occurring in in-door environments. The primary function of this assistant is to track context and provide pertinent information facilitating humans to accomplish tasks during these events. Since provision of pertinent information serves as a memory aid to humans, we conveniently call this ubiquitous computing service 'Memory Jog' (MJ). The MJ resembles a context-aware conference assistance [25] and has been selected for studying computing services based on implicitly derived information in the scope of the CHIL project.

4.2. Distributed multi-agent implementation of the Memory Jog

The MJ service was implemented in the smart room depicted in Fig. 2, which consists of:

- One 64 channel microphone array [26].
- Microphones for localization, in particular three clusters, each consisting of four microphones.
- Four fixed cameras, used for overall monitoring of the room.
- One active camera with pan, tilt and zoom (PTZ camera).
- A panoramic (or fish-eye) surveillance camera.

The service implementation takes advantage of the middleware infrastructure depicted in Fig. 4. At the lowest level of this infrastructure, perceptual components process sensor streams. To this end, middleware capturing data from all available sensors has been produced. Captured data are made available for processing in any of the systems, based on the distributed NIST Smartflow middleware (NSFS). Hence, the NSFS system constitutes the solution adopted for high performance transport of streams.

Perceptual processing of sensor data is based on the following components technologies that have been developed in our lab:

- Acoustic identification and localization of the speaker [27].
- Face Detection, Recognition and People tracking [28,22].
- Detection of speech activity.

Perceptual processing is computationally demanding. Therefore, perceptual components are implemented in low-level high performance languages (i.e., C/C++), and wrapped as JADE agents in line with the notion illustrated in Fig. 3. Wrapping was implemented through a perceptual components wrapper agent, as shown in Fig. 9. Accordingly, we combined perceptual components in order to create higher level perceptual components that can track situations as illustrated in Fig. 5. Fig. 10 depicts how elementary components tracking the agenda, identifying speech activity, identifying faces and recognizing people are used to form composite perceptual components that keep track of the status of a whiteboard and the meeting room table, with respect to the meeting participants. The higher level '*Table Watcher*' and '*White-board Watcher*' perceptual components are accordingly use to track situations.

The situations to be tracked are driven by the situation model depicted in Fig. 6. This situation model is loaded in the Situation Watching Agent based on an XML file describing the model. This XML format specifies the perceptual components output combinations leading to detecting a particular situation. These combinations are also described in Table 1, which specifies the perceptual components values that determine the transition to each one of the contextual states of the situation model.

The subscription mechanism illustrated in Fig. 7, was exploited to detect situations and triggering the service logic. The service logic of the MJ service was based on a wide range of services offered within the smart room. Smart room services were implemented based on the introduced sensor, services and actuator control framework. Specifically, the following services were implemented based on this framework: (a) a TTS (Text-to-Speech) service, (b) a slide show/display service and (c) a storage service allowing access to a relational database. These services were invoked by the service logic, either based on current context, or upon end users' requests. The latter requests can be issued

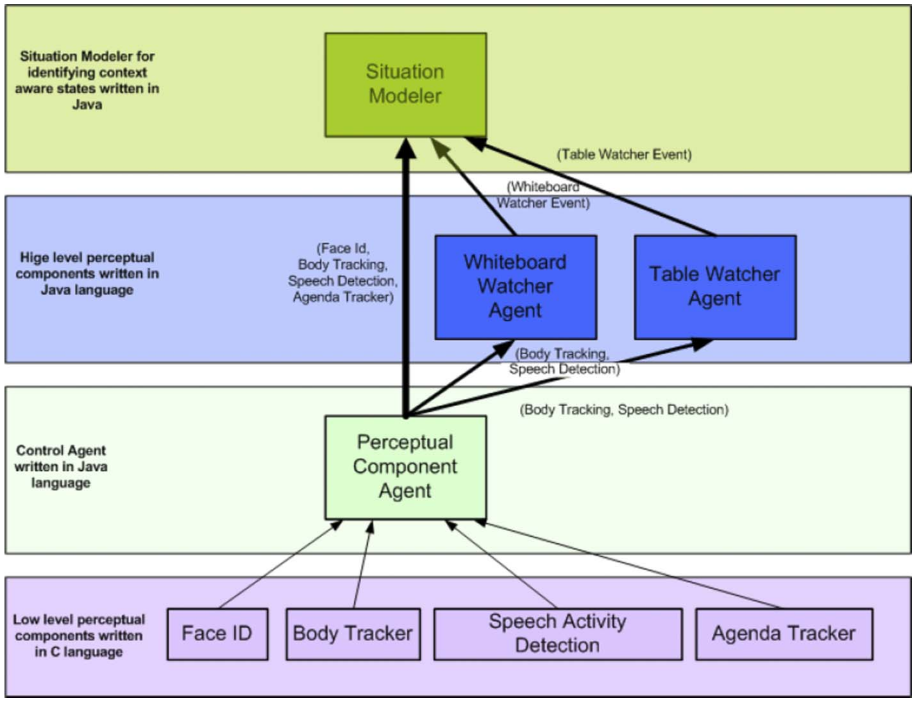

Fig. 10. Perceptual components supporting the situation model.

Table 1
Identifying composite contextual states through combining perceptual components' outputs

Situation transition	Combinations of perceptual components outputs
NIL $\rightarrow$ S1	TableWatcher = N (N people in table area) and Speech Activity (SAD = true)
S1 $\rightarrow$ S2	WhiteboardWatcher = 1 (1 person in speaker area), TableWatcher = $N - 1$ ($N - 1$ people in table area)
S2 $\rightarrow$ S3	Acoustic Localization (Table Area)
S3 $\rightarrow$ S2	Acoustic Localization (Speaker Area)
S2 $\rightarrow$ S4	WhiteboardWatcher = 0 (no person in speaker area), TableWatcher = N (N people in table area)
S4 $\rightarrow$ S5	TableWatcher = 0 (no people in table area)

by a graphical user interface that is used by the end user to visualize MJ information, as well as to allow interaction with the smart room. A snapshot of this interface is provided in Fig. 11.

4.3. Autonomic features

All these agent types extend the same agent class, which realizes transparent communication capabilities, subscriptions, poling, as well as agent discovery and communication. In order to ensure the autonomic functionality of the overall system, we augmented our basic JADE agent class with additional failure detection and healing functionality as outlined in the previous section.

Autonomic functionality was implemented as an additional layer over the basic JADE functionality (Fig. 12). Therefore, all agent types described above we endowed with healing capabilities since they were based on the same augmented version of JADE agent. Therefore, autonomy constitutes a vertical pillar of all distributed entities.

4.4. Users evaluation

The overall implementation of the MJ service confronted a host of technical challenges as outlined above. Apart from technical challenges however, ubiquitous services need to take into account user issues, with a view to ensuring that services are appealing to end users. User acceptance is a hot issue for non-obtrusive services, given that the vast majority of end users are not acquainted with the emerging context-aware computing paradigm.

In order to evaluate the MJ service prototype in terms of user acceptance, we performed two simulations studies. In each case one potential end-user ('the subject') was asked to use the MJ service along with members of the design team who played as actors in the scenario. The subject of the study was well briefed on the CHIL project, the particular scenario and the background to the MJ service. Members of the design team configured the scenarios and made observations relating to the end user's behavior. Upon completion of each simulated scenario the user was interviewed to gain feedback about the service usability, the

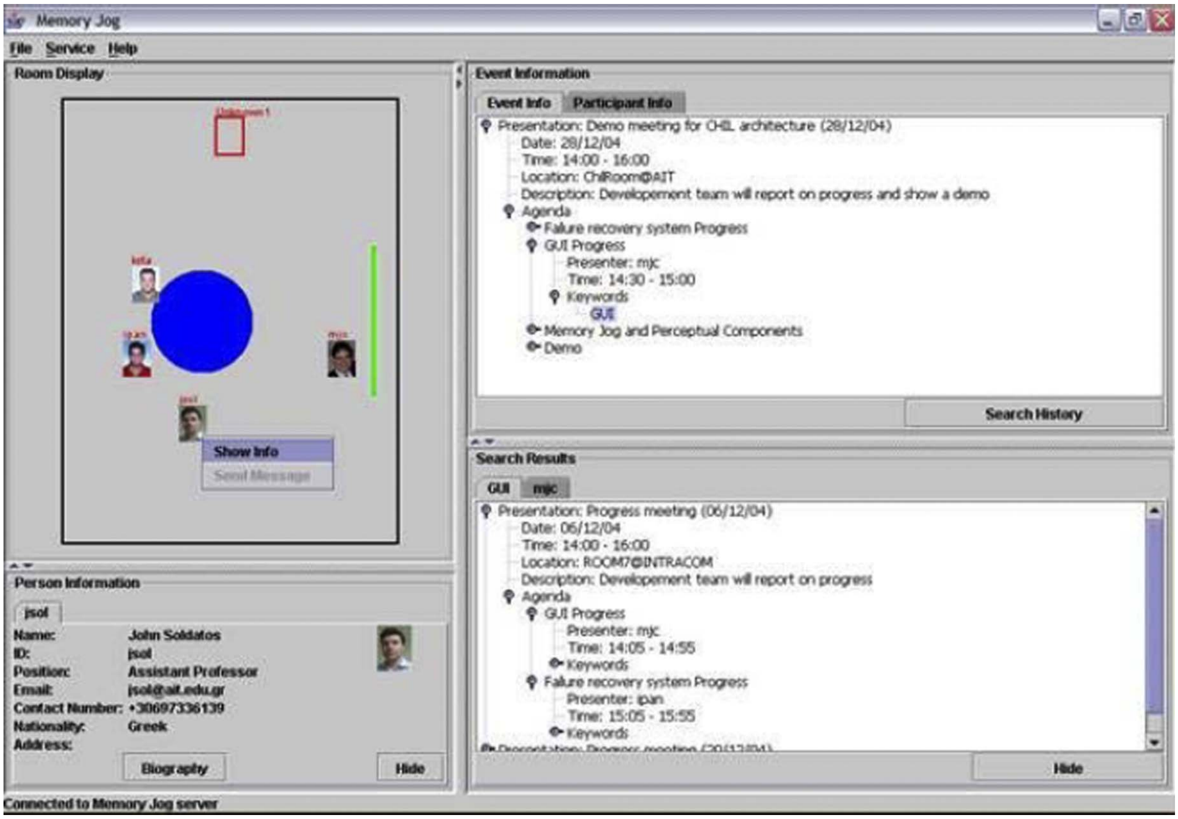

Fig. 11. A snapshot of the Memory Jog user interface.

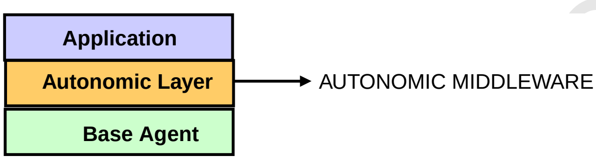

Fig. 12. Enhancing software agents with autonomic functionality.

potential impact of the service, as well as to get ideas about possible improvements and additions to the service.

Both scenarios occurred in the room depicted in Fig. 2. The first scenario involved a simple presentation with two participants. The purpose of this scenario was to provide a fictional example according to which the initial service prototype was designed. As such the context is not necessarily one that would occur in the real world. It involves two developers in the room and one developer who is a virtual participant and is not physically present in the room. The presentations were about recent work that each of the participants has done.

The second scenario involved a meeting, where participants aimed at reporting the progress of their work on a project. In particular, this involved a regular progress meeting of four developers and a project manager, in which each member of the team presents the latest developments in their work. A member of the administration of the institution also attends the meeting as new hardware and software needs to be purchased and they are responsible for approving the purchase. The regular meeting is also used to address high-level project management a system design issues.

The simulation studies revealed several issues with respect to the service prototype. As far as the intuitiveness of the user interface is concerned, the interface was generally perceived as being user friendly. Also, the information displayed was easily understood and accessed. However, users declared that certain features (e.g., the search button) need to be made more obvious.

With respect to the level of interaction and obtrusiveness of the MJ service, the current functionality was perceived not to be overly intrusive in that it did not require a great deal of user input to respond to and adapt to the progress of the event. Recommendations were made, however, to make changes more obvious when they occur. A suggested way to achieve this was the use of interactive timed pop-up info boxes.

End users suggested also additional functionalities, such as the ability to view the presentation slides through the interface, and to possibly make personal annotations on the slides. Moreover, users asked for pop-up or audio reminders about the timing of the event, so that speakers are reminded of when their time is running out. More trivial recommendations concerned showing a reminder at the end of the event as to when the next event in a sequence is scheduled, as well as to who should attend. These suggestions will be seriously taken into account into designing the next version of the MJ service.

5. Conclusions

Middleware architectures boost rapid application development in the scope of complex and heterogeneous network and computing infrastructures. The increasingly important role of middleware components is intensified, when it comes to addressing ubiquitous computing applications and services. Middleware infrastructures for such applications impose a need for balancing between transparency and context-awareness, while at the same time tackling with more sophisticated environments in terms of hardware and software. Furthermore, a ubiquitous computing application asks for a wide range of runtime services such as context-awareness, sensor streams capturing, transfer and processing, dynamic service discovery and invocation, as well as autonomic capabilities.

In supporting these features, a host of middleware components have to be implemented and integrated. Agent platforms provide a sound foundation for implementing such runtime services in a distributed environment. In this paper we have introduced an agent based middleware framework, which can ease the implementation of sophisticated context-aware services in appropriately configured in-door environments (called ‘smart rooms’). Smart rooms comprise a rich set of video and acoustic sensors, enabling several perceptive interfaces to operate and provide elementary context cues. The introduced agent framework provides functionality for service access control, personalization, context modeling, as well as of dynamic control and management of sensors and actuating devices. Context modeling relies on the network of situations approach, which allows composite contextual states to be detected and tracked based on a combination of perceptual components outputs. The sensor and actuator control framework, allows ubiquitous computing services to dynamically access information on the status of infrastructure elements, as well as to invoke their services. Moreover, the agent framework has been augmented with fault tolerance capabilities ensuring that failures are timely detected and restored.

Based on this agent framework, we have implemented the Memory Jog, an intelligent non-obtrusive service providing pertinent information and assistance in the scope of meetings, seminars and presentations. This implementation has leveraged the capabilities of the agent based framework, therefore allowing the service developer to focus on the service logic implementation rather than the middleware. In implementing this service we have taken advantage of a wide range of perceptive interfaces including face detection, face recognition, person tracking (both visual and acoustic), as well as speech activity detection. The corresponding perceptual components have been used to trigger a simple situation model, which has been encoded into the agent framework. Following situation detection, the Memory Jog leverages the sensor, service and actuator control framework to invoke TTS services, display services, as well as storage services allowing the retrieval of past information relating to the current contextual state.

Acknowledgements

This work is part of the FP6 CHIL project (FP6-506909), partially funded by the European Commission under the Information Society Technology (IST) program. The authors acknowledge valuable help and contributions from all partners of the project, especially from participants in Work-package 2 dealing with the architecture and software infrastructure supporting the CHIL services. Special thanks also to Michael Carras, Dr. Pnevmatikakis and Dr. Talantzis for their valuable contribution in setting up and configuring our smart room infrastructure.

References

[1] G. Tesauro, D.M. Chess, W.E. Walsh, R. Das, A. Segal, I. Whalley, J.O. Kephart, S.R. White, A multi-agent systems approach to autonomic computing, in: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04), vol. 1, New York City, New York, USA, July 19–23, 2004, pp. 464–471.

[2] S.S. Yau, F. Karim, Y. Wang, B. Wang, S.K.S. Gupta, Reconfigurable context-sensitive middleware for pervasive computing, in: IEEE Pervasive Computing, joint special issue with IEEE Personal Communications on Context-A ware Pervasive Computing, 1(3), July–September 2002, IEEE Computer Society Press, Los Alamitos, USA, pp. 33–40.

[3] M. Weiser, The Computer for the 21st Century, in: Scientific American, vol. 265, no. 3, 1991, pp. 66–75.

[4] A.K. Dey, D. Salber, G.D. Abowd, A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, in: Human-Computer Interaction 16, 2001.

[5] A.K. Dey, Understanding and using context, in: Personal and Ubiquitous Computing Journal, vol. 5(1), 2001, pp. 4–7.

[6] A. Murphy, G. Picco, G.-C. Roman, LIME: a middleware for physical and logical mobility, in: Proceedings of the 21st International Conference in Distributed Computing Systems, IEEE CS Press, Los Alamitos, CA, 2001, pp. 524–533.

[7] B. Johanson, A. Fox, T. Winograd, ‘The interactive workspaces project: experiences with ubiquitous computing rooms’, IEEE Pervasive Comput. Mag. 1 (2) (2002) 67–75.

[8] D. Garlan, D. Siewiorek, A. Smailagic, P. Steenkiste, Project Aura: Towards distraction-free pervasive computing, IEEE Pervasive Comput., 2002, pp. 22–31.

[9] U. Saif, H. Pham, J.M. Paluska, J. Waterman, C. Terman, S. Ward, A case for goal-oriented programming semantics, in: System Support for Ubiquitous Computing Workshop at the 5th Annual Conference on Ubiquitous Computing (UbiComp '03), 2003, pp. 74–83.

[10] M. Coen, B. Phillips, N. Warshawsky, L. Weisman, S. Peters, P. Finin, Meeting the computational needs of intelligent environments: the metaglue system, in: 1st International Workshop on Managing Interactions in Smart Environments (MANSE'99), Dublin, Ireland, December 1999, pp. 201–212.

[11] B. Brumitt, J. Krumm, B. Meyers, S. Shafer, Ubiquitous computing and the role of geometry, in: IEEE Pers. Commun., 2000, pp. 41–43.

[12] V. Stanford, Pervasive computing and smart work spaces: integration, interoperability and interfaces, in: Pervasive Computing 2000, New IT Industry Conference, January 25–26, NIST, USA, 2000.

[13] O. Galibert, C.Martin, M. Michel, F. Mougin, V. Stanford, The NIST smart space data flow modular test bed – an environment for interoperability, in: IAB Meeting, Rutgers CAIP Center, September 13, 2000.

[14] The CHIL project. <http://chil.server.de>.

[15] S.R. Ponnekanti, B. Johanson, E. Kiciman, A. Fox, portability, extensibility and robustness in iROS, in: Proceedings of IEEE International Conference on Pervasive Computing and Communications (Percom 2003), Dallas-Fort Worth, TX, March 2003, pp. 11–19.

[16] S. Shafer, J. Krumm, B. Brumitt, B. Meyers, M. Czerwinski, D. Robbing, The new easy living project at microsoft research, in: DARPA/NIST Workshop on Smart Spaces, July 1998, pp. 127–130.
[17] N. Minar, M. Gray, O. Roup, R. Krikorian, P. Maes, Hive: distributed agents for networking things, in: IEEE Concurrency 8(2), April–June 2000, pp. 24–33.
[18] T. Hammond, K. Gajos, R. Davis, H. Shrobe, An agent-based system for capturing and indexing software design meetings, in: Proceedings of the International Workshop On Agents in Design – WAID'02. Cambridge, MA, August 2002.
[19] H.-C. Wong, K. Sycara. A taxonomy of middle-agents for the Internet, in: Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS'2000), July, 2000, pp. 465–466.
[20] Java Agent Development Environment. <http://jade.tilab.com/>.
[21] FIPA – The Foundation for Intelligent Physical Agents. <http://www.fipa.org>.
[22] J. Soldatos, L. Polymenakos, A. Pnevmatikakis, F. Talantzis, K. Stamatis, M. Carras, Perceptual interfaces and distributed agents supporting ubiquitous computing services, in: Proceedings of the Eurescom Summit 2005, April 2005, pp. 43–50.
[23] A. Paar, J. Reuter, J. Schaeffer, J. Soldatos, I. Pandis, M. Carras, A Pluggable Architectural model and a programming language independent API for an ontological knowledge base server, in: The 4th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE), Agia Napa, Cyprus, October 31–November 4, 2005.
[24] J.L. Crowley, Context driven observation of human activity, in: Proceedings of the European Symposium on Ambient Intelligence, October 2003.
[25] A.K. Dey, M. Futakawa, D. Salber, G.D. Abowd, The Conference Assistant: Combining context-awareness with wearable computing, in: Proceedings of the 3rd IEEE International Symposium on Wearable Computers (ISWC'99), San Francisco, CA, IEEE, October 20–21, 1999, pp. 21–28.
[26] M. Brandstein, D. Ward, Microphone Arrays: Techniques and Applications, Springer-Verlag, New York, 2001.
[27] F. Talantzis, A.G. Constantinides, L. Polymenakos, Estimation of direction of arrival using information theory, in: IEEE Signal Processing Letters, August 2005, to appear.
[28] A. Pnevmatikakis, L. Polymenakos, Comparison of eigenface-based feature vectors under different impairments, in: Proceedings of the 17th International Conference on Pattern Recognition (ICPR 2004) (1), pp. 296–299.

John K. Soldatos, was born in Athens, Greece in 1973. He obtained his Dipl-Eng. degree in 1996 and his PhD in 2000, both from the Electrical and Computer Engineering Department of the National Technical University of Athens (NTUA). He has had an active role in several EU co-funded research projects (EXPERT AC-094, WATT AC-235, IMPACT AC-324, Chameleon EP 20597, CATCH-2004 IST-1999-11103, and LION IST-19990-11387), and is now involved in the CHIL-FP6-506909 project. He has also consulted in many ICT projects for leading Greek enterprises (INTRACOM S.A, PEGASUS S.A, IBM Hellas S.A, OTE S.A, TEMAGON S.A). Dr. Soldatos has extensively lectured in NTUA and AIT, while he has also given numerous invited lectures. As a result of his activities he has co-authored more than 60 papers published in international journals and conference proceedings. Since March 2003 he is with Athens Information Technology, where he is currently an Assistant Professor. His current research interests are in Pervasive/Ubiquitous and Autonomic Computing, Grid Computing and Broadband Traffic Control.

Ippokratis Pandis is a Ph.D. candidate at Carnegie Mellon University (CMU). Before joining CMU, Ippokratis was member of the Autonomic and Grid Computing research group of Athens Information Technology (AIT), where he worked for the CHIL-FP6-506909 project. Ippokratis has been interested and holds publications to several conferences in the areas of database systems, middleware for ubiquitous computing and hypertext/hypermedia systems. Ippokratis got his B.Sc. from the Computer Engineering and Informatics Department (CEID) of the University of Patras, Greece, and his M.Sc. from the Information Networking Institute (INI) of the Carnegie Mellon University.

Konstantinos Stamatis, obtained his Master of Engineering degree in 2002 from the National Technical University of Athens and his Master of Science degree in Athens Information Techology, after the completion of MSIN program of Carnegie Mellon University. Mr. Stamatis has knowledge of C, C++ and Java programming languages, experience in implementing software applications, as well as in Unified Modeling Language and Soft Systems Methodology. His current research interests lie in the fields of hardware design and ubiquitous/pervasive computing systems. He is currently involved in the CHIL-FP6-506909 project.

Lazaros C. Polymenakos, obtained his electrical engineering and computer science degree from the National Technical University of Athens Greece (1989), and his Masters (1991) and Doctoral degrees (1995) from the Massachusetts Institute of Technology, Cambridge, MA, USA. Since 1995 he has worked with the IBM Human Language Technologies group on robust methods for automatic speech recognition in the presence of external noise and in varying channels. He has worked on methods for improving accuracy in speech recognition in embedded devices, and is the author of several technical publications, presentations and patents. In 1996 he was visiting professor at Rutgers University, NJ, USA. In 1998, he joined IBM Hellas, SA where he has focused on research and development for the Greek speech recognition system. Since 2002, he has been a faculty member of AIT, where he focuses on research in signal processing, perceptual interfaces and distributed systems.

James L. Crowley, directs the GRAVIR laboratory (CNRS UMR 5527) at the INRIA Rhone-Alpes research center in Montbonnot (near Grenoble), France. He holds the post of Professor at the Institut National Polytechnique de Grenoble (INPG), where he teaches courses in Computer Vision, Signal Processing, Pattern Recognition and Artificial Intelligence at l'ENSIMAG (Ecole National Superieure d'Informatique et de Mathematiques Appliquèes). Professor Crowley has edited two books, five special issues of journals, and authored over 180 articles on computer vision and mobile robotics. He ranks number 1466 in the CiteSeers most cited authors in Computer Science (July 2004).