An Ontology-based Framework for Dynamic Resource Management in Ubiquitous Computing Environments

Ippokratis Pandis, John Soldatos,
Athens Information Technology (AIT)
19,5Km Markopoulo Avenue, GR-19002, Peania, Athens, Greece
{ipan, jsol}@ait.edu.gr

Alexander Paar, Jürgen Reuter
Institute for Program Structures and Data Organization
Am Fasanengarten 5, 76128 Karlsruhe, Germany
{alexpaar, reuter}@ipd.uka.de

Michael Carras, Lazaros Polymenakos
Athens Information Technology (AIT)
19,5Km Markopoulo Avenue, GR-19002, Peania, Athens, Greece
{mjc, lcp}@ait.edu.gr

Abstract

Ubiquitous computing applications are supported by sophisticated middleware components enabling dynamic discovery, invocation and management of resources, as well as reasoning in cases of uncertainty. This paper advocates Semantic Web technologies as primary vehicles to achieve dynamic management of resources in ubiquitous computing infrastructures and services. We introduce a framework for implementing ubiquitous computing services comprising a large number of sensors and perceptive interfaces, emphasizing the role of knowledge bases for dynamic registration and invocation of resources. We present the use of ontology-based mechanisms for controlling sensors and actuators. Moreover, we describe the implementation of a knowledge base server that can leverage different ontology management systems, while also exposing a host to different client access interfaces. The introduced framework has been exploited in implementing real prototype ubiquitous computing services, which we also outline in the paper.

1. Introduction

The ubiquitous computing vision [1] proposes an essential paradigm shift in the way we use computers. Among the core characteristics of ubiquitous computing services is context awareness. Ubiquitous computing applications derive contextual information based on a rich collection of casually accessible, often invisible sensors that are connected to a network structure. Another characteristic of ubiquitous computing environments is that they are dynamic, since sensors, devices, components and services may dynamically join or leave [2]. In order to support context-awareness and dynamism, ubiquitous computing environments comprise a wide range of components including middleware for controlling sensors and actuating devices, perceptual components extracting context from sensor streams and information fusion components combining context towards identifying complex contextual states.

Both middleware and hardware components are characterized by extreme diversity in terms of functionality, underlying technologies and vendors. Managing heterogeneity is crucial to facilitating application development. In this paper we present an architectural framework for ubiquitous computing services, emphasizing on the mechanisms for dynamic discovery, invocation and management of resources (i.e. services, devices, networks, sensors, actuators, perceptual and information fusion components). This framework has been developed to support several non-obtrusive services developed in the Computers in the Human Interaction Loop (CHIL) project [6-7]. CHIL deals with a large number of perceptual components (e.g., 2D-visual components, 3D-visual perceptual components, acoustic components, audio-visual components, as well as output components). These components are developed by several groups at four different smart spaces (also called 'smart rooms'). The diversity of technology components, the complexity of the services, as well as the number of collaborating sites, pose integration challenges that existing middleware architectures (e.g., [3-5]) fail to adequately address. The framework proposed in this paper addresses these challenges based on Semantic Web technologies such as ontologies. Ontologies provide semantic power in discovering resources and have therefore added value over conventional registries (e.g., UDDI [8]). Moreover, they support reasoning,

which is a prerequisite to building services using diverse perceptual components.

The rest of the paper has the following structure: Section 2 presents an overview of the structuring principles of ubiquitous computing services developed within CHIL. It presents the usage of ontologies and focuses on the part of the architecture dealing with dynamic management of resources based on ontologies. Section 3, elaborates on the structure and the implementation of the ontology and the knowledge base. Section 4, describes prototypical applications that exploit the overall architecture framework. Finally, section 5 draws basic conclusions.

2. Overview of Ontology Based Architectural Framework

2.1. Core Agent Framework

Our core agent framework (see also [2]) boosts the reusability of components, and establishes a common methodology for the development of services in a ubiquitous computing environment. This framework: (a) provides a set of basic services within each smart room, like mechanisms to control the various sensors and actuators in a common way, and subscription mechanisms for the monitoring of the state of the system, (b) allows augmentation and evolution of the underlying infrastructure independently of the services installed in the smart room, (c) controls user access to services, (d) enables integration of new services, based on a 'pluggable' mechanism.

Figure 1 depicts an anatomy of this multi-agent framework. The various agents of this framework can be classified in core agents, basic service agents and ubiquitous service agents. Core agents are service and smart room installation independent. They provide a communication mechanism for the distributed entities of the system, a set of basic services regarding the control of the installed infrastructure, while also allow service providers to 'plug' service logic into the framework. Core agents include the Device Agent, the Personal Agent, the Situation Watching Agent, the Smart Room Agent, the Knowledge Base Agent and the Agent Manager.

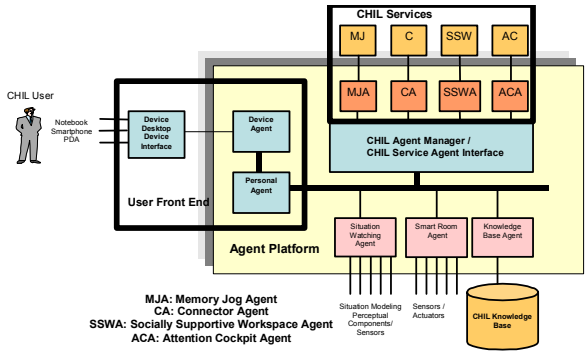

Figure 1: Multi-agent system facilitating ubiquitous computing applications.

The Device Agent adapts the device of the user (e.g., PDAs, Smart Phone) to the rest of the system. The Personal Agent maintains the user profile and personalizes the services. The Situation Watching Agent keeps track of contextual states. It offers a subscription and a corresponding notification mechanism, towards notifying interested agents of contextual changes. The Smart Room Agent acts as a proxy to the various infrastructure-specific services of the system. The Knowledge Base Agent is a wrapper of the interface of the implemented Knowledge Base Server that exposes it to the agent platform. Finally, the Agent Manager acts as a matchmaker. In case of a service request (from another agent), it discovers through the knowledge base which agent can solve the problem and what resources are needed for this.

Basic Services Agents are plugged to the Core Agents, while also exposing their capabilities through the ontology management system. Two bundles of basic services are supported: (a) Context awareness services based on identification of situations (according to the network of situations approach [7]) and (b) Services offered by sensors, actuators and other devices of the smart room [2]. Core Agents wrap the corresponding actual implementers of services and allow them to cooperate with the rest of the system. Ubiquitous service agents implement the non-obtrusive intelligent service logic of the various context-aware services. Each service is therefore implemented as a Ubiquitous service agent and plugged into the framework. A set of services is currently being implemented and integrated into the framework through the following agents: a Memory Jog Agent (MJA), a Connector Agent (CA), a Socially Supportive Workspaces Agent (SSWA), and an Attention Cockpit Agent (ACA).

2.2. Role of the Ontologies

At the heart of the presented agent framework is an ontology management mechanism that:

- Provides a registration mechanism for the framework that acts as a yellow pages registry for the discovery, manipulation, and integration of needed resources of the system, such as sensors, perceptual components, situation models and context-aware services.
- Provides a permanent storage solution for the data of the applications in a platform-, operating system- and programming language- independent way.
- Enables the interchangeability between the various developed perceptual components. With the adoption of a common ontology all the participants in the project have a common vocabulary for the perceptual components they develop. Two perceptual components (developed by different vendors), which provide the same functionality, may be interchangeable if they deliver the information in the same language. Therefore, the perceptual components share their knowledge and the ontology is used to annotate data and functionality which is to be shared between them.
- Facilitates the communication between the various agents of the system. Part of the ontology is used to model both the type and value of the content of the messages which are exchanged among the various agents of the system. Thus, the same interoperability as well as interchangeability benefits can be achieved as for perceptual components.

2.2.1. Registration Mechanisms

Automatic discovery, binding and invocation of the required components of a ubiquitous computing application can greatly facilitate the portability of services across different environments, while also relaxing the effort required for configuring, administering and operating such a complex distributed system. Thus, we envisage a kind of 'yellow pages' service towards dynamically discovering and invoking the components that offer the context-awareness, required for a specific service. The ontology management mechanism is used as a registration mechanism in a four-fold way, as depicted in figure 2. In particular, with the use of the developed ontology for the system, we can register:

Installation-specific components: Sensors, devices and actuators are registered with the ontology following their installation. Registration information includes elements about vendors, models, interfaces, capabilities, network addresses, as well as the means to access their capabilities (i.e. their API).

Perceptual components: Perceptual component providers install and configure perceptual components processing sensorial input. Given that the registry contains information about sensors, actuators, their physical and logical configuration, etc., the developer of perceptual components consults this information towards configuring and registering its components. APIs for registering the perceptual components to the directory services are provided. The various perceptual components are categorized according to their inputs and outputs, based on appropriate ontological concepts.

Situation Models: The situation model developer, who builds situation models according to the network of situations approach [7], obtains descriptions of perceptual components and infrastructure elements, in order to model context states. According to the network of situations approach context cues can be combined towards identifying higher level contextual states. Situation models are also registered to the directory service, along with its pointers to underlying sensors, actuators and perceptual components. During run-time parsing of a situation model these components are dynamically discovered and invoked.

Services: Context-aware service logic is based on service actions that are initiated when contextual changes are identified. Service actions are defined and registered in the directory service. Entire context-aware services are also registered to facilitate their activation and exploitation by other services.

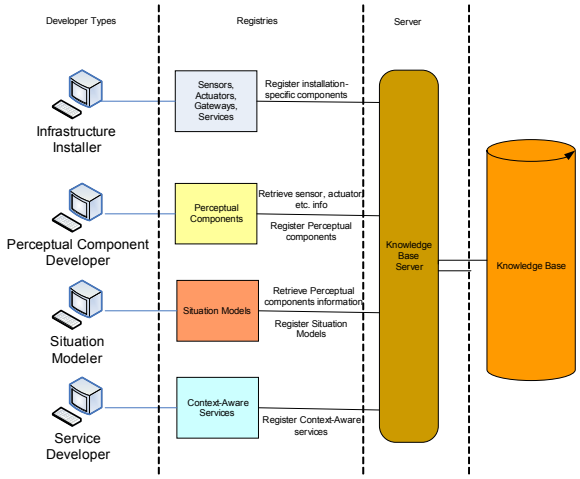

Figure 2: Registration mechanism with the usage of ontologies as a horizontal element of the implemented framework.

2.2.2. Sensor /Actuator Control

Part of the ontology is dedicated to sensor and actuator information (e.g., vendor, model, current status of the device, interfaces, capabilities, network addresses). This information is accessible to other agents of the system through the KBA. Monitoring and control of sensors and actuators is performed through the special proxy agents that represent the sensor or actuator in the world of agents. Proxy agents, register the device to the knowledge base, upon bootstrapping of the device. Moreover, they update the operational state of the device in the registry whenever its status changes. Proxy agents provide information regarding the sensor/actuator state (i.e. to other agents), while also enabling other agents to execute control commands on them. Control commands might be required with a view to regulating the environment. Note that proxy agents provide a degree of abstraction: each proxy exposes a universal virtualized interface to the agent framework (i.e. a common interface for sensors/actuators of the same type). A sensor-actuator specific driver is required to adapt the universal interface commands to the low-level capabilities of each particular sensor or actuator. This low-level driver is based on the control API offered by the sensor or actuator (e.g., IEEE1394 camera). We have actually implemented three proxy agents: one generic, one for microphones and one for cameras.

2.2.3. Smart Room Services

The ontology includes high-level descriptions of value-adding services installed in the infrastructure (e.g., text-to-speech (TTS) services, display services). A mechanism for controlling smart room services has been also implemented, similarly to the one described for the control of the sensors and actuators. 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. All the services that provide the same functionality (e.g.,. all the TTS services) are wrapped by a service proxy of this type (e.g,. a TTS proxy). Hence, all the requests for such a service are being forwarded to the proxy of that service, which determines the specific implementation that will handle the request. The proxy retrieves (from the knowledge base) run-time information about the properties and the operational status of the actual services and then takes the decision.

The algorithm for deciding which service is the most appropriate for a request depends on the proxy implementation. For example, in the case of four different operational TTS service instances, the TTS service proxy would decide to forward the request to the machine that it is nearer to the service target, while a display or targeted-audio service proxy would decide to forward the request to the machine 'opposite' to the target.

Figure 3 illustrates the mechanism enabling discovery of (actuating) services. The proxy of a service registers to the system (step A). All the providers of this specific service also register themselves into the system (step B). Whenever a client makes a request for a service invocation, it sends this request to the gateway for all the services (step 1), which is the Smart Room Agent (SRA). The SRA searches the registry for a proxy for such a service (step 2) and forwards the request to it (step 3). When the service proxy receives a request, it checks the registry for available service providers (step 4), and activates a selection algorithm to decide to which provider to forward the request (step 5). All the searches in the knowledge base are performed at run-time, to allow for service providers to dynamically join/leave the system.

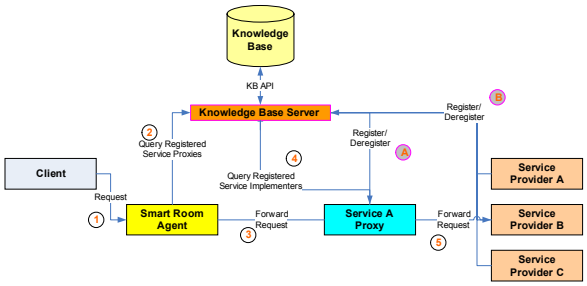

Figure 3: Dynamic discovery and invocation mechanism for actuating services.

The following section takes a more technical and detailed look into the design of the ontology that is currently being used in the context of the CHIL research project, as well as, into the implementation of the knowledge base and its corresponding server software.

3. The CHIL Ontology and Knowledge Base Implementation

Ontologies use description logic [9] to hierarchically classify and describe entities and relations between them. In implementing an ontology prototype we selected the Web Ontology Language OWL [10] as a standard technology for inter-component and inter-host communication. Besides the ontology itself, we implemented a Knowledge Base Server, which manages ontologies and ontology based information.

3.1. Overall Structure of the CHIL Ontology

The CHIL ontology strives to establish a general-purpose core vocabulary (i.e. conceptual knowledge) for the various concepts within a multi-sensor smart space and context-aware applications executed within it. In order to minimize the complexity of the overall ontology, we modularized the ontology allowing different parts to be used in different contexts and applications. The `owl:imports` statement of OWL perfectly meets the goal of modularization, and allows modules to live in a namespace of their own. In this way, name collisions can be avoided. Separated namespaces are ideal for collaborative work: developers may safely introduce new concepts locally in their module's name-space without interfering with other modules. Assuming that other modules use similar concepts which should be merged, the core module may provide a merged version of the concept. To globally put together all the modules, the ontology consists of a main OWL file, which imports all modules. Developers interested only in a subset of modules can define a main OWL file of their own that imports only the modules of interest.

Besides the main OWL file, the ontology consists of the core module `chil-core`, a communication agent module `chil-ca`, and an example module `chil-isl`; more are likely to be added in the future. The former two modules contain conceptual knowledge, also known as terminological- or TBox knowledge. This kind of knowledge provides the vocabulary that is required to describe an environment or a situation. The actual description is realized with assertional- or ABox knowledge. The `chil-isl` module prototypically illustrates the setup of a smart room in terms of the conceptual knowledge of the other modules.

3.2. CHIL Ontology for Perceptual Components and Situation Modeling

A major purpose of the ontology is to facilitate the lookup of Perceptual Components (PCs) by describing what kind of information they provide. The CHIL ontology provides the vocabulary to describe the kind of information a PC can deliver (e.g. coordinates of a person). Accordingly, the `chil-core` module introduces concepts of perceivable entities such as for example `Person`, `MeetingRoom`, `Table` or `Whiteboard`, as well as perceivable roles of such entities, e.g. the `Location` of a `Person` or the `ActivityLevel` of a `MeetingRoom`. PCs require and provide data in manifold formats and on a range of abstraction levels, not only on the high level of concepts and roles that we just described. The ontology provides a rudimentary set of vocabulary on data formats such as audio or video streams. In this way, PCs can be functionally characterized and looked up by their interface input and output data types, regardless of their level of abstraction and internal structure.

The ontology describes also situations and actions towards situation modeling. For modeling chronological arrangement of events, we follow James F. Allen's temporal logic [11]. Assuming that precise points in time can not be modeled anyways, we instead base the chronological relation of events on intervals that may touch, overlap, be included, or stay side by side without touching each other. Allen's temporal logic rises to its full power when applying the set of first-order logical rules that Allen presents. This is currently not possible with plain OWL. However, there are several approaches underway which are aiming at adding rules to OWL (e.g., [12]). Currently, rule-based reasoning itself is done outside of the ontological framework. Still, however, the ontological framework can store ontological data that an external rule-based reasoner can access.

3.2.3. Ontological Reasoning with OWL

Ontologies are used to model the knowledge of an application domain in a structured and formally well-understood way. There are algorithms that provide sound and complete reasoning for the description logic $\mathcal{SHIQ}$ [13]. Reasoning is important to ensure the quality of an ontology and to fully exploit its rich structure. Though we do not rely on ontological reasoning in the scope of our applications, our overall architecture supports a future implementation. Following we give three examples that introduce the OWL features inverse properties, transitive properties, and nominals.

Inverse Properties: If an OWL property `P1` is tagged as the `owl:inverseOf` OWL property `P2`, then for all `x` and `y`: `P1(x,y) iff P2(y,x)`. Let `Meeting` and `Person` be two OWL classes and `Meeting hasParticipant Person` and `Person attendsMeeting Meeting` be two OWL properties where `hasParticipant` is declared to be the `owl:inverseOf` of `attendsMeeting`. Further, let `Person(JIM)` be an instance of `Person` and `Meeting(CHIL_MEETING)` be an instance of `Meeting`. Given the fact that `JIM attends-Meeting CHIL_MEETING`, calling the method *listPropertyValuesOfIndividual*("`hasParticipant`", "`CHIL_MEETING`") of the CHIL Knowledge Base Server, which returns all OWL instances that are values of the specified OWL property of the specified

OWL instance, would yield `JIM` although one has never explicitly stated this fact. Instead, it is computed by the OWL reasoner based on the fact that the OWL property `hasParticipant` is the `owl:inverseOf` of the OWL property `attendsMeeting`.

Transitive Properties If an OWL property `P` is specified as transitive, then for any `x`, `y`, and `z`: `P(x,z) if P(x,y) and P(y,z)`. Let the OWL classes `MeetingRoom`, `MeetingVenue`, and `Person` be subsumed by `LocatableThing` and let `LocatableThing locatedIn LocatableThing` be declared as a transitive property. Given the facts `MeetingRoom(ROOM3)`, `MeetingVenue(HOTELADLON)`, `Person(JIM)`, and `JIM locatedIn ROOM3`, calling the CHIL Knowledge Base Server method *listIndividuals* ("`Person`", "`locatedIn`", "`HOTELADLON`"), which returns all OWL instances of the specified type whose specified OWL property value equals the specified value, would yield `JIM`, although one has never explicitly said that `JIM` is `locatedIn HOTELADLON`.

Nominals: Nominals are special class names that are to be treated as singleton sets. Given the classes `FaceRecognizer`, `DoorObserver` (both subsumed by `PerceptualComponent`), `Door` (subsumed by `LocatableThing`), and the property `PerceptualComponent pointsTo LocatableThing`, one could express that a `PerceptualComponent` which `pointsTo` a `Door` is a `DoorObserver`. Given the facts `FaceRecognizer(MY_RECO)`, `Door(DOOR3)`, and `MY_RECO pointsTo DOOR3`, calling the CHIL Knowledge Base Server method *listIndividualsOfClass* ("`DoorObserver`"), which returns all OWL instances that have the specified OWL class among their types, would yield `MY_RECO` although `MY_RECO` has never been explicitly declared as a `DoorObserver`.

3.3 The CHIL Knowledge Base Server

Despite the rise of OWL as the lingua franca for the Semantic Web, there is still a lack of commonly used APIs and widely accepted tools for processing ontological information. Managing OWL data is still error prone and laborious. Also, most off-the-shelf ontology management systems lack connectivity, which makes it difficult to connect to these systems using programming languages that are not natively supported. Widely used ontology management systems (e.g., [13-15]) can only be accessed locally within a Java software environment. There is no common way for remote access via for example Java RMI. On the other hand, there are programming language agnostic remoting APIs such as DIG [16]. Nevertheless, these APIs provide limited support for error- and exception handling, as well as for preserving the consistency of managed ontologies. Moreover, these APIs are not specifically tailored to OWL.

In the scope of our knowledge base, we developed an adapter for off-the-shelf ontology management systems that makes it easy to benefit from readily available inferencing technology, while also facilitating connectivity to several programming languages. This software adapter is referred to as CHIL Knowledge Base Server. This server is accessible both locally and remotely through a unique interface. Since client components in a pervasive computing environment may be written in a variety of programming languages, the remote interface is programming language independent. Also internal and external data representations are architecture independent to avoid interoperability problems (e.g. byte order, little-/big endian number representations). The server can cope well with multiple, potentially competing incoming requests in parallel (i.e. thread safe server design). Finally, the Knowledge Base Server API is tailored to OWL.

Figure 4 depicts the architectural model of the CHIL Knowledge Base Server. Emphasis was put on providing flexible connectivity capabilities and to make it easy to use different underlying ontology management systems based on a uniform client API. Design Patterns and Java reflection were used in order to plug in hosts for arbitrary remoting protocols dynamically at runtime and to replace the underlying ontology management system and/or its persistency policies.

The interface `IOntology` defines an API for editing and querying OWL ontologies. Currently, 150 methods are approximately defined to tell and ask the ABox and TBox of an OWL DL based ontology [17]. Off-the-shelf ontology management systems are wrapped by `IOntology` implementations. In Figure 4 the class OntologyJena2 would adapt to HP Labs' Jena 2 framework. The interface `IKnowledgeBase` extends not only `IOntology` but also `java.rmi.Remote` to identify it as an interface whose methods can be invoked from remote. Thus, the Java RMI port is different from other remoting hosts which are to be implemented as extensions of the abstract base class `KnowledgeBaseServer` that defines methods to bind and unbind the respective remoting host. These server implementations are aggregated by `KnowledgeBaseServerHost` and

refer to the singleton instance of `KnowledgeBase`. This instance handles all incoming requests from Java RMI and all aggregated remote hosts. Persistence capabilities are defined by the `IPersistence` interface inheriting to `IKnowledgeBase`. It is possible to either use built in persistence capabilities of the underlying ontology management system or to extend these features. Thus, one can combine off-the-shelf ontology inference-, manipulation-, and persistence features from a variety of systems.

The Knowledge Base Server was implemented as an Eclipse plug-in and is hosted by instances of the Eclipse workbench (http://www.eclipse.org). A "*New Knowledge Base Wizard*" lets one chose the underlying ontology management system and the remoting protocols that are to be hosted by this instance of the Knowledge Base Server. The wizard creates adapters for ontology management systems and remoting protocol hosts. Thus, it is possible to dynamically deploy extensions to the Knowledge Base Server. Ontology management system adapters are available for HP Labs' Jena 2 and IBM's Semantic Network Base. Beyond Java RMI, an XML-over-TCP API was implemented to provide programming language independent access to the Knowledge Base Server. Client libraries are available for C#, C++, and Python.

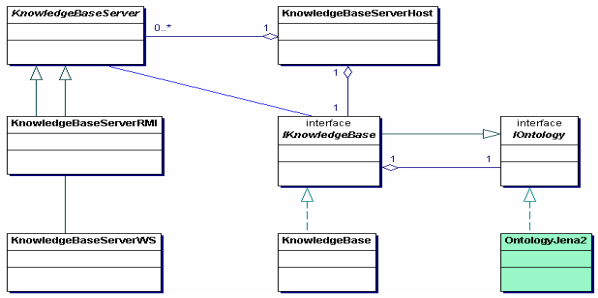

Figure 4: Architectural model of the CHIL Knowledge Base Server.

4. Proof-of-Concept Applications

The goal of the Memory Jog (MJ) service [2, 7] is to provide pertinent non-obtrusive information in the scope of meetings, lectures and presentations. The service is designed to automatically provide contextual information (i.e. in a non intrusive manner), but also to respond to user requests (i.e. in an on-demand mode). The MJ is built according to the overall framework depicted in Figure 1, and makes extensive use of the ontology. In particular, it uses the ontology as a registry towards discovering localization and tracking components (i.e. visual person trackers and acoustic localizers). The use of the ontology has greatly facilitated the task of interfacing service logic components to perceptual components. Moreover, the MJ relies on the sensor, actuator and service control frameworks to access the capabilities of output services (e.g., TTS). Based on these frameworks, MJ developers focus on the service logic and are decoupled from the tasks of querying and testing service availability. Moreover, these frameworks are used to provide add-value services, such as the intelligent display service.

The Intelligent Display service provides streams for display to the most 'appropriate' display device based on contextual information. This service exploits the capabilities of the service control framework that has been developed. The service subscribes to the context-acquisition system (i.e. through the Situation Modeling Agent) and thereafter gets continuous notifications about the position and the orientation of the various actors. Using the ontology-based registry the service gets information about operational display devices. Accordingly, ontological reasoning is used to provide an optimal uninterrupted display of streams, messages, presentations, etc. When the targeted user changes his position the service is notified and (if needs be) the stream is transferred from one device to another.

5. Conclusions and Future Work

Ubiquitous computing services comprise a wide range of distributed heterogeneous hardware, software and middleware components. Establishing structuring principles for integrating these components is a key prerequisite to easing the implementation and increasing the penetration of emerging ubiquitous computing applications. All major ubiquitous computing projects have established sophisticated middleware architectures, which however do not account for the challenges of integrating a large number of perceptual and information fusion components over multi-senor infrastructures. These challenges relate to the establishment of intelligent methods for discovering, invoking and managing distributed components, as well as to the availability of reasoning mechanisms in cases of uncertainty. Emerging semantic web technologies have the potential to successfully tackle these issues.

In the scope of the CHIL project we have worked out an architectural framework for ubiquitous computing services in the scope of in-door environments. This framework relies on a knowledge base for dynamically discovering and invoking components. It is therefore appropriate for dealing with the dynamic nature of ubiquitous computing applications, where sensors and devices are likely to

dynamically join or leave the environment. We have developed a sensor and actuator control framework, which underpins some of the values of the CHIL ontology.

In implementing the knowledge base server we have accounted for the need to support different ontology management systems, as well as for the need to enabling client access based on a variety of interfaces. To this end, appropriate adapter modules have been implemented to different ontology management systems, along with utilities automatically generating client code.

The use of the knowledge base server in the scope of real prototype ubiquitous computing applications has manifested the benefits of the ontology middleware in building and deploying applications. It is within our plans to use the ontology to register the whole range of middleware components used in MJ. We also intend to leverage the reasoning capabilities of the ontology.

6. Acknowledgments

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.

7. References

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

[2] Soldatos, J., Pandis, I., Stamatis, K., Polymenakos, L., and Crowley, J. L. (2004). Agent Based Middleware Infrastructure for Autonomous Context-Aware Computing Services. Accepted for publication in the Journal of Computer Communications, Elsevier.

[3] Johanson, B., Fox, A., Winograd, T. (2002). The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. In IEEE Pervasive Computing Magazine 1(2), April-June 2002.

[4] Garlan, D., Siewiorek, D., Smailagic, A., and Steenkiste, P. (2002). Project Aura: Towards distraction-free pervasive computing. In IEEE Pervasive Computing, pp. 22–31, 2002.

[5] Saif, U., Pham, H., Paluska, J. M., Waterman, J., Terman, C., Ward, S. (2003). A Case for Goal-oriented Programming Semantics. In System Support for Ubiquitous Computing Workshop at the 5th Annual Conference on Ubiquitous Computing (UbiComp '03).

[6] The Computer in the Human Interaction Loop (CHIL) project. At address: http://chil.server.de.

[7] Soldatos, J., Polymenakos, L., Pnevmatikakis, A., Talantzis, F., Stamatis K., and Carras M. (2005). Perceptual Interfaces and Distributed Agents supporting Ubiquitous Computing Services. In the Proceedings of the Eurescom Summit 2005, April 2005, pp. 43-50.

[8] Bellwood, T. et al. (2003). The Universal Description, Discovery and Integration (UDDI) specification. Version 3, 2003. At address: http://www.uddi.org/.

[9] Bader, F., Calvanese, D., McGuiness, D., Nardi, D., Patel-Schneider, P. (2003). The Description Logic Handbook, Cambridge University Press, 2003.

[10] W3C Recommendation: OWL Web Ontology Language Overview (2004). At address: http://www.w3.org/TR/owl-features/.

[11] Allen, J.F. (1984). Towards a general theory of action and time. In Artificial Intelligence, vol..23(2):123-154.

[12] W3C Member Submission: SWRL (2004). A Semantic Web Rule Language Combining OWL and RuleML. At address: http://www.w3.org/Submission/SWRL/.

[13] HP Labs: Jena 2 - A Semantic Web Framework (2004). At address: http://www.hpl.hp.com/semweb/jena.htm.

[14] Stanford University School of Medicine: Protégé knowledge acquisition system (2003). At address: http://protege.stanford.edu/.

[15] IBM Alphaworks: Snobase - IBM Ontology Management System (2003). http://alphaworks.ibm.com/tech/snobase.

[16] Bechhofer, S. (2003). The DIG Description Logic Interface: DIG/1.1, University of Manchester.

[17] W3C Recommendation: OWL Web Ontology Language Guide (2004). At address: http://www.w3.org/TR/2004/REC-owl-guide-20040210/.