Use of Ontologies in Pervasive Computing Environments

Robert E. McGrath, Anand Ranganathan, Roy H. Campbell, M. Dennis Mickunas

Department of Computer Science
University of Illinois, Urbana-Champaign
Urbana, Illinois

Report number: UIUCDCS-R-2003-2332 UILU-ENG-2003-1719

April, 2003

This research is supported in part by the National Science Foundation grant NSF 98-70736, NSF 9970139, and NSF infrastructure grant NSF EIA 99-72884.

Ontologies in Pervasive Computing Environments

Contents

Ontologies in Pervasive Computing Environments

Abstract

Pervasive Computing Environments consist of a large number of independent entities that help transform physical spaces into computationally active and intelligent spaces. These entities could be devices, applications, services or users. In recent years, advances in middleware have enabled the different entities to interact with each other. However, it is still difficult to assure that independent entities can understand the "semantics" of the environment and other entities when they interact with each other. To tackle this problem, we have used semantic web technologies to attach semantics to various concepts in Pervasive Environments. We have developed ontologies to describe different aspects of these environments. Ontologies have been used to make information systems more usable. They allow different entities to have a common understanding of various terms and concepts and smoothen their interaction. They enable semantic discovery of entities, allowing requests to be semantically matched with advertisements. The ontologies also describe the different kinds of operations an entity supports like asking queries and sending commands. This makes it easier for autonomous entities to interact with one another. It also allows the generation of intelligent user interfaces that allow humans to interact with these entities easily. The ontologies also allow external agents (such as new entities that enter the environment or entities that access the environment over the web) to easily interact with the environment. Finally, we use ontologies coupled with description logic to ensure that the system is always in a consistent state. This helps the system meet various security and safety constraints. We have incorporated the use of ontologies in our framework for pervasive computing, GAIA [63]. While we have used ontologies in the pervasive computing scenario, many of the issues tackled are applicable to any distributed system or multi-agent system.

Use of Ontologies in Pervasive Computing Environments

1. Introduction

Pervasive (or Ubiquitous) Computing Environments are physical environments saturated with computing and communication, yet gracefully integrated with human users [39]. These environments advocate the construction of massively distributed computing environments that feature a large number of autonomous entities (or agents). These entities could be devices, applications, services, databases or users. Various types of middleware (based on CORBA, Java RMI, SOAP, etc.) have been developed that enable communication between different entities. However, existing middleware have no facilities to ensure semantic interoperability between the different entities. Since different entities are autonomous, it is infeasible to expect all of them to attach the same semantics to different concepts on their own. In order to enable semantic interoperability between different entities, we take recourse to methods used in the Semantic Web [6, 83].

When two autonomous entities exchange messages they must have common interfaces and protocols, including a common message format. In addition, the parties must know or discover the semantics of the messages: the vocabulary of the messages, which includes the names and valid values of message elements. Essentially, the parties must have a shared schema for interpreting the messages. *Semantic interoperability* is the establishment of shared schemas for exchanging messages.

1.1. Semantic Interoperability in Pervasive Computing Systems

Open distributed systems face a fundamental challenge: autonomous entities (e.g., independent producers and consumers) need to successfully exchange messages containing descriptions of entities, services, events, and other concepts. The descriptions take several forms:

- Advertisement: send a description of the offered service or interface to the Registry or other receiver
- Notification: send a description of an event (e.g., arrival of an entity)
- Query: send a description of the desired entities or services, receive a set of descriptions of entities or services

In each case, the sender and receiver must be able to interpret the contents of the messages, i.e., they must know the schema.

For a simple or closed system, all the required schemas are (implicitly or explicitly) compiled into the components. But in an open system, the parties are autonomous, heterogeneous, and evolving. In this situation, it is necessary to be able to discover and use schemas as needed, as the system runs.

The driving problems require new techniques for:

- Schema exchange – open publishing and discovery of schemas
- Composing schemas – combining schemas from autonomous sources
- "Semantic Query" – queries and notification using more than simple key word matching

Thus, besides the message protocols and syntax, the messages must have a machine-readable schema. In this open system, there must be an open model for defining, exchanging, and using schemas.

1.1.1. Object Registries are Insufficient

Object registries, such as the CORBA Naming Service [51] and the RMI Registry [68], provide a basic mechanism for finding well-known (i.e., known in advance) services. Brokers, such as the CORBA Trading Service [53], provide the capability to locate services by attributes. Many other services provide similar features, including LDAP [96], JINI [13], and Microsoft's Registry [56].

These standards define the interfaces and formats for descriptions, but they leave most of the specific content—the "semantics" of the descriptions—to publishers, communities, and applications developers. The communities must create and share standard schemas for across the open system: this process is typically outside the definition of the registry.

For example, the CORBA Trading Service is a standard interface for a broker, but does not define the properties of the advertised services, or the legal values of properties [53]. By design, the specification of valid properties and relationships is left to communities, such as the CORBA Domain Task Forces [52]. The CORBA Trading Service lacks a number of important features, such as:

- The matching rules are limited, and only explicitly defined attributes can be matched
- The Service Type (schema) are limited and weakly enforced on Service Offers (advertisements). For example, any property from any (or no) Service Type can be used in a Service Offer or query.
- Relations between Service Types and objects are minimally supported (e.g., by properties), and are not enforced by the Trading Service.

Finally, while the queries have a grammar, there is no declarative language for defining Service Types (schema) or Service Offers (instances). The contents of the Trading Service are defined by API calls (i.e., code), which are not easy to publish and maintain in an open system.

Similarly, the JINI Discovery Service defines an architecture, protocols, and interfaces for advertisement, notification, and discovery [69]. Services and objects are described by Service Entry objects, which are sets of attributes. But, as in the case of CORBA, the attributes of objects and the legal values of properties are left to applications and communities. The JINI Discovery Service also does not define a standard schema definition language, or standard mechanisms for managing or validating Service Entry objects.

Chakraborty et al. [8] report an augmented JINI registry, DReggie, which is similar to our approach. They use DAML+OIL to define schemas for objects that are registered with JINI.

1.1.2. Web Services are Insufficient

In recent years, the Web Services architecture has emerged as a set of standards for publishing, discovering, and composing independent services in an open network [87, 89]. Industry (e.g., Microsoft .NET [28, 47], IBM WebSphere [36]) and the Global Grid Forum "Grid Services" [74] are built on top of the Web Services architecture. This architecture seeks to improve electronic commerce by enabling open markets using networks and Web services [88, 92]. A key goal of the Web Services architecture is "matchmaking", or mutual discovery by producers and consumers [22, 73, 87].

The Web Services architecture is an abstract framework which defines the problems, generic architecture, and general approach [87, 88]. Essentially, the Web Services architecture is a "virtualization" of services, include a generic registry, the UDDI [76]. There may be different technical realizations of this architecture, but the current work has focused on a solutions based on XML, which may be implemented with any underlying database or registry.

The message passing protocol uses SOAP [84], the content of the messages is delivered in the Web Services Description Language (WSDL) [89, 90]. WSDL is a language for describing a network connection which responds to certain messages (a **portType**) [89]. WSDL does not specify how **portTypes** are discovered.

In the current Web Services technology, discovery is implemented with the Universal Description, Discovery, and Integration (UDDI) specification [75, 76]. The UDDI's "purpose is the representation of data and metadata about Web services" ([76], p. 19). The UDDI defines a standard registry and protocols, designed for, but certainly not limited to, business-related Web Services. This specification defines a language and APIs for publishing descriptions of network services, including their protocol and interface requirements.

The UDDI descriptions can use vocabularies from many classification and identification schemes. The schemes include official standards such as United Nations Standard Products and Services Code (UNSPSC) [77] or application specific classification scheme or vocabulary. The UDDI descriptions use "technical models" (**tModels**), which are a reference to a specification of the scheme used in the description, i.e., the schema ([76], p. 46). The UDDI defines a few standard **tModels**, but most **tModels** are references to specifications outside UDDI, i.e., a pointer to some other standard. Furthermore, the

UDDI does not define the format or description language for **tModels**. These are left to publishers ([76], p. 47), and many are paper documents. Thus, each **tModel** may be implemented differently, and must be programmed case by case.

The Web Services architecture attempts to meet the challenges of service discovery, including the need to manage descriptions of services from multiple autonomous sources [88, 91, 92]. But the Web Services standards have not yet defined the "semantic" layer—standards for specifying, validating, and exchanging schemas. The Semantic Web is designed to fill this role (e.g., [1, 7, 60, 73]), but the integration of Semantic Web and Web Services has yet to be specified.

1.1.3. Context-Aware Applications

A successful ubiquitous computing environment must have context-aware behavior. Context plays a huge role in ubiquitous environments – applications in pervasive and mobile environments need to be context-aware so that they can adapt themselves to rapidly changing situations. Applications in ubiquitous environments use different kinds of context information, such as location of people, activities of individuals or groups, weather information, etc.. Distributed infrastructure, such as the Context Toolkit [12] provide useful middleware for constructing context-aware applications. The infrastructure must also manage context information and provide mechanisms to ensure that the different entities that use context have the same semantic understanding of contextual information.

There are different types of contexts that can be used by applications. These include physical contexts (location and time), environmental contexts (weather, light and sound levels), informational contexts (stock quotes, sports scores), personal contexts (health, mood, schedule, activity), social contexts (group activity, social relationships, whom one is in a room with), application contexts (email, websites visited) and system contexts (network traffic, status of printers).

We represent contexts as predicates. The structure of the context predicate depends on the type of context. The infrastructure must define the vocabulary and types of arguments that may be used in the predicates. The various types of contextual information that can be used in the environment must be well-defined so that different entities have a common understanding of context. Also, there must to be mechanisms for humans to specify how different applications and services should behave in different contexts.

Context-aware applications need to discover and interpret aspects of the environment relevant to their goals, the "vocabulary" of the context. Thus, context-aware applications need semantic services as well. We use the same Ontology Server for ontologies for context information.

A lot of work has been done in the area of context-aware computing in the past few years. However, not much effort has been spent in developing ontologies for context information.

Seminal work has been done by Anind Dey, et al. in defining context-aware computing, identifying what kind of support was required for building context aware applications and developing an infrastructure that enabled rapid prototyping of context-aware applications [12] . While the Context Toolkit does provide a starting point for applications to make use of contextual information, it does not provide much help in organizing the wide range of possible contexts in some structured format. It also doesn't provide ways of defining the different kinds of contexts available to applications.

Ontologies have been used in Multi-Agent Systems. MyCampus [64], which is an agent-based environment for context-aware mobile services uses ontologies for describing contextual attributes, user preferences and web services, making it easy to accomodate new task-specific agents and web services. It, however, does not make use of reasoning mechanisms to ensure logical consistency of the ontologies.

Rcal [59] is a Distributed Meeting Scheduling software that negotiates meeting times based on user's availability and preferences. RCal can reason about schedules published on semantic web (written in RDF, based on some ontology) and automatically incorporate them in user's schedules.

The RETSINA Multi-Agent System Infrastructure [70] uses ontologies based on WordNet to enable mappings between similar words or synonyms. This allows agents to communicate with each other more effectively.

Tamma, et al. [71] describes the use of ontologies to enable automated negotiation between agents. The ontologies used describe various terms used in the negotiation process.

1.1.4. A Missing Piece: Adding "Semantics" to the Distributed Object System

Interoperability, context-awareness, and discovery face similar fundamental challenges: enabling autonomous entities (e.g., independent producers and consumers) to exchange messages successfully. These messages may contain descriptions of entities, services, events, context information, and other concepts.

In a discovery protocol, the messages take several forms:

- *Advertisement:* send a description of the offered service or interface to a Registry or another receiver
- *Notification:* send a description of an event (e.g., arrival of an entity that matches a subscription)
- *Query:* send a description of the desired entities, services or other concepts, receive a set of descriptions of entities or services or other concepts.

When two or more autonomous entities (e.g., users, agents, services, or applications) interact, they need to determine the interfaces and protocols required to communicate with the other parties. The interacting entities must share a common set of terms and concepts on which their interaction can be based. Descriptions and messages required for this operation include:

- *Service/interface description:* A description of the interfaces provided by components and services
- *Notification message:* a description of an event (e.g., service availability)
- *Negotiation messages:* an exchange of messages describing proposed interfaces, services or other concepts, offered or requested

A context-aware application must obtain current context information, from whatever sources are available in the current space. Context events are implicitly or explicitly descriptions of changes in environmental conditions. Context-aware services require meta-level information: definitions of the types of contexts that can be used and their structures. The context will be used in several operations, including:

- *Discovery*: Entities discover sources of context information
- *Notification of context events:* When environmental conditions change, certain entities must be notified
- *Interrogation:* Entities can request a description of certain context information and they receive descriptions of the context of the current space.

In each case, the sender and receiver must be able to interpret the contents of the messages, i.e., they must know the schema of the messages.

If these services may be viewed as one or more abstract database, it is clear that an additional component is needed, namely the schema or information model which defines the structure for the contents of the database, queries, and notifications. Basically, the entities of the ubiquitous computing environment need to know or determine:

- What objects exist
- What their attributes are
- What questions can be asked
- What answers may be returned
- What the questions and answers mean

Standard schemas are needed to describe many kinds of entities, including people, places, and things. Furthermore, the system has policies, constraints, and relationships which may need to be discovered as well. For a robust system, it is necessary to have a flexible mechanism for exchanging descriptive information of many kinds.

For a simple or closed system, all the required schemas are compiled into the components (implicitly or explicitly). But, in an open system, the parties are autonomous, heterogeneous, and evolving. In this situation, it is necessary to be able to discover and use schemas as needed, as the system runs. To do this, besides the message protocols and syntax, the messages must have a machine readable schema. In this open system, there must be an open model for defining, exchanging, and using schemas. This requires new techniques for:

- Schema exchange – open publishing and discovery of schemas
- Composing schemas – combining schemas from autonomous sources, e.g., two different vendors
- "Semantic Query" – queries and notification other than key word matching (i.e., semantic equivalence between concepts rather than just syntactic equivalence)

The "Semantic Web" is designed to address these challenges for the World Wide Web. This study applies Semantic Web technology to Ubiquitous Computing.

1.2. The Semantic Web

The so-called "Semantic Web" is a set of emerging technologies mostly adopted from earlier work on intelligent agents [6, 83]. The essence of the Semantic Web is a set of technology-independent, open standards for the exchange of descriptions of entities and relationships [14, 17, 27, 40, 46, 50] This includes XML-based languages and formal models for Knowledge Bases. While the "Semantic Web" was designed to enhance Web search and agents, we show that it is well suited to some of the requirements of a ubicomp system.

In this study, *ontologies* written in DAML+OIL XML documents [9] to describe various parts of the GAIA environment. An Ontology Server manages a system ontology and operations on DAML ontologies. The ontologies are loaded into a Knowledge Base (KB), built on the FaCT Server [4, 32]. The KB implements automated reasoning algorithms to prove the ontology is consistent with the KB, and to answer logical queries about the KB.

An ontology is a formal vocabulary. Ontologies establish a joint terminology between members of a community of interest. These members can be humans or automated agents. The DAML+OIL provides a language to share ontologies via XML documents, and the Ontology Server provides a common interface for using the ontologies.

Each entity in our environment uses the vocabulary and concepts defined in one or more ontologies. When two different entities talk to each other, they know which ontology the other entity uses and can thus understand the semantics of what the other entity is saying. The use of Semantic Web technologies to describe these environments also allows web-based entities to access and interact with these environments.

1.3. Semantic Infrastructure for Ubiquitous Computing: An Experimental Implementation

Ontologies can be used for for describing various concepts in a Pervasive Computing Environment. We have developed ontologies that describe the different kinds of entities and their properties. These ontologies define different kinds of applications, services, devices, users, data sources and other entities. They also describe various relations between the different entities and establish axioms on the properties of these entities (written in DAML) that must always be satisfied.

We have an ontology that describes the different types of contextual information in GAIA. Context plays a huge role in pervasive environments – applications in pervasive and mobile environments need to be context-aware so that they can adapt themselves to rapidly changing situations. Applications in pervasive environments use different kinds of contexts (like location of people, activities of individuals or groups, weather information, etc.).

The ontologies that describe the pervasive environment greatly help in the smooth operation of the environment. Some of the ways in which we use ontologies in our pervasive environment are:

- Checking to see if the descriptions of different entities are consistent with the axioms defined in the ontology. This also helps ensuring that certain security and safety constraints are met by the environment
- Enabling semantic discovery of entities
- Allowing users to gain a better understanding of the environment and how different pieces relate to each other
- Allowing both humans and automated agents to perform searches on different components easily
- Allowing both humans and automated agents to interact with different entities easily (say, by sending them various commands)
- Allowing both humans and automated agents to specify rules for context-sensitive behavior of different entities easily
- Enabling new entities (which follow different ontologies) to interact with the system easily. Providing ways for ontology interoperability also allows different pervasive environments to interact with one another.

In this report, we describe how ontologies have been incorporated in our pervasive computing environment, GAIA. Section 2 describes the different kinds of ontologies we have in our system. Section 3 gives details on some of the ways in which we use ontologies to ease the interaction between different entities in the system. Section 4 gives some implementation details. Section 5 describes our experiences with using ontologies. Section 6 describes related work in the field and Section 7 concludes the paper.

2. Background

We have integrated semantic services into our prototype Pervasive Computing Environment, GAIA. The infrastructure provides a standard and ubiquitous service that may be used by any entity of the system. In particular, the infrastructure enables any entity of the system to use of ontologies written in standard DAML+OIL XML [9]. The infrastructure maintains a system ontology and Knowledge Base that integrates knowledge about the software, hardware, environment, and physical entities of the Pervasive Computing Environment.

Ontologies are used for describing various concepts in the GAIA Environment. We have developed ontologies that describe the different kinds of entities and their properties. The ontologies define different kinds of applications, services, devices, users, data sources and other entities. They also describe various relations between the different entities and establish axioms on the properties of these entities that must always be satisfied.

Another important use of ontologies is to describe different types of contextual information in GAIA. The ontology defines standard descriptions for locations, activities, weather information, and other information that may be used by context-aware applications.

2.1. GAIA: a Pervasive Computing Environment

GAIA is our infrastructure for Smart Spaces, which are ubiquitous computing environments that encompass physical spaces [63]. GAIA converts physical spaces and the ubiquitous computing devices they contain into a programmable computing system. It offers services to manage and program a space and its associated state. GAIA is similar to traditional operating systems in that it manages the tasks common to all applications built for physical spaces. Each space is self-contained, but may interact with other spaces. GAIA provides core services, including events, entity presence (devices, users and services), discovery and naming. By specifying well-defined interfaces to services, applications may be built in a generic way so that they are able to run in arbitrary active spaces. The core services are started through a bootstrap protocol that starts the GAIA infrastructure. starts the GAIA infrastructure. GAIA uses CORBA to enable distributed entities to communicate with one another.

We have used GAIA to manage rooms in our Computer Science building. GAIA helps make these rooms smart and responsive to the needs of different users. There are a wide variety of devices that exist in these rooms. These include authentication devices like fingerprint sensors and smart card readers, display devices like large plasma screens, video walls, handheld devices, wearable devices like smart watches and smart rings, various input devices like touch screens and microphones, etc. Besides, there are large number of applications and services like music-playing applications, presentation applications and drawing applications. Ontologies provide an easy way to manage this diversity in our environments.

GAIA has served as our test-bed for the use of ontologies in ubiquitous computing environments. We have incorporated the use of ontologies and Semantic Web technology into the GAIA infrastructure, to provide semantic services for applications, services, and users of the GAIA environment. The implementation is described in section 5 below.

2.2. Semantic Web Technology

The so-called "Semantic Web" is a set of emerging technologies based on the Web standard XML [6, 83], but based on the Web standard XML. The essence of the Semantic Web is a set of technology independent, open standards for the exchange of descriptions of entities and relationships using XML-based languages and formal models for

Knowledge Bases [14, 15, 17, 50]. While the Semantic Web technology was developed to support Web search [15, 45, 50] and intelligent agents [27, 44, 46, 73], it turns out to be well suited to the requirements of a ubicomp system, as will be shown below. This section briefly introduces the Semantic Web.

Semantic Web technology is a set of standards for *open exchange of resource descriptions*. In the Web community, a "resource" is a generic term for any document, object, or service that can be accessed via the WWW. The objects and services of a ubicomp system can be considered to be instances of such resources.

The Semantic Web has developed technology for managing standard descriptive vocabularies, *ontologies*. The DAML+OIL specification provides an XML standard format for ontologies, which makes it much easier to publish, import, and reuse standard vocabularies, and to create specialized vocabularies. Furthermore, DAML+OIL are mapped to a formal logical model which can be used to ensure logical consistency and to answer logic queries including *satisfiability*, *equivalence*, and *subsumption* [17, 25, 26, 32].

Together, these technologies support the creation and formal validation of ontologies for specific domains, which can be combined into larger systems. The general approach is for relatively small groups of "experts" to develop small, tractable vocabularies for specific topics, presumably reusing and extending more general standard vocabularies. The domain experts represent the key concepts and relations in a formal vocabulary which is formally verified and published in an XML file.

An application or service may well use components, services, or other entities from several domains, each of which may have an ontology defined for it. These domain-specific ontologies can then be imported into a larger system as needed, i.e., as the domain becomes relevant. The combined ontology needs to be validated, and correspondences between different vocabularies (e.g., multiple terms for the same concept) must be recognized and used. Also, the combined vocabularies may create

Table 1. Information Models for Discovery

Technology	**Schema or Vocabulary**	**Model**
CORBA	Service Type	Left to communities [51], e.g., CORBA Domain Task Forces [50]
JINI	ServiceInfo	Not Specified ([66])
UDDI (ebXML, etc.)	tModel, Classification	Left to publishers ([73], p. 47), e.g., OASIS Technical Committees [49]
WSDL (.NET, Grid, etc.)	serviceDescripton, portType	Not specified ([86, 87])
Semantic Web	Ontology (RDF, DAML+OIL, OWL)	Description Logic specification [25, 26]

implicit relationships not stated in any of the individual vocabularies. The DAML+OIL XML language is specifically designed to meet these requirements.

The Semantic Web fills an important gap in distributed object technology. Conventional distributed object systems and emerging standards such as Web Services define interfaces and protocols for registries, advertising, and discovery, but do not provide standards for defining the content of these services, i.e., descriptions of entities. The ontologies of the Semantic Web fill this role

Table 1 lists several distributed system standards which describe entities. Each provides a mechanism to specify a classification or vocabulary to be used as a schema for the contents of the descriptions used in the service, but a formal model and language, services must rely on manual coding. The Semantic Web standard provides ontologies that may be used by any of these standards.

2.3.1. The Semantic Web Stack: XML, RDF, DAML+OIL, OWL, and Description Logic

The Semantic Web standards of interest include the XML languages which, with their formal underpinnings, are designed to be an open language for exchanging information between Knowledge Bases. This section reviews the Semantic Web standards "stack" (Figure 1).

The World Wide Web standards provide a universal address space (URIs [5]), and the XML language is a universal standard for content markup. The XML standard assures a universal (and multilingual) namespace and syntax ([78, 80, 94, 95]): an XML document is guaranteed to be parseable, but there is no constraint on how to interpret the tokens. The same information can be encoded in many ways using XML.

The Resource Description Framework (RDF) defines an XML language for expressing entity-relationship diagrams [82]. Essentially, RDF defines standard tags for expressing a network of related objects. However, RDF does not specify a single logical model of entities or relationships: the same relationship could be encoded in many ways. XML and RDF are necessary but not sufficient for the exchange of complex information in open systems. The additional requirement is one or more standard logical models, to constrain the use and interpretation of tags.

The DARPA Agent Markup Language (DAML) and Ontology Interchange Language (OIL) are XML languages (combined as DAML+OIL) are designed to provide the required models. The OIL is a language for describing formal vocabularies (ontologies), essentially a meta-format for schemas [17, 30, 32]. The DAML is a language for describing entity-relationship diagrams that conform to a schema (i.e., an OIL ontology) [2, 24-26]. The DAML+OIL XML language will be standardized for the Web as the Web Ontology Language (OWL) [79, 81, 85, 86, 93]. For most purposes, OWL will be identical to DAML+OIL.

The DAML+OIL language is an XML binding to a formal logical model. Specifically, DAML+OIL is bound to a *Description Logic* [19, 32]. *Description Logics* are a general class of logic are specifically designed to model vocabularies [11, 19, 32]. Unlike most XML documents, a DAML+OIL XML document is a set of statements in a formal logic: essentially the DAML+OIL language uses the mechanisms of XML to deliver *well-defined logic programs*. Therefore, unlike XML and RDF alone, a DAML+OIL document has a single, universal interpretation. While there may be many ways to express the same idea in DAML+OIL, a given DAML+OIL document has only one correct interpretation.

The DAML+OIL language, with its formal underpinnings, is designed to be an open language for exchanging information between Knowledge Bases. A Knowledge Base (KB) is a database augmented with automated reasoning capabilities. A KB not only answers queries by match, it also can deduce results using automated reasoning. The automated reasoning also can maintain the consistency of the KB as new information is added or modified.

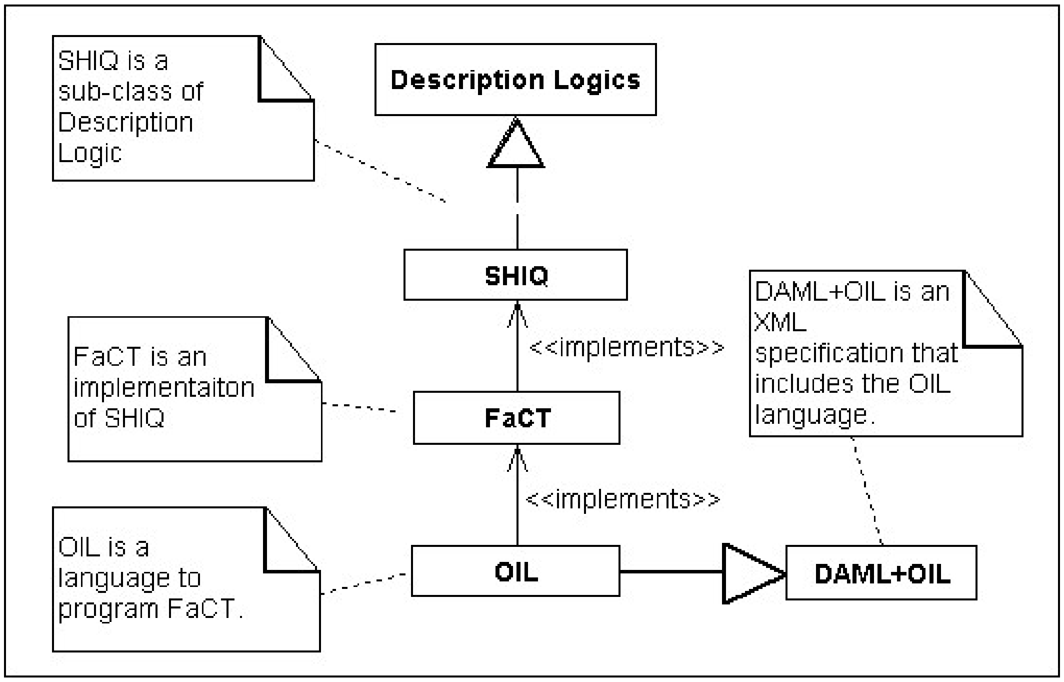

SHIQ is a sub-class of Description Logic
Description Logics
SHIQ
DAML+OIL is an XML specification that includes the OIL language.
<<implements>>
FaCT is an implementaiton of SHIQ
FaCT
<<implements>>
OIL is a language to program FaCT.
OIL
DAML+OIL

Figure 2. The logic, logic language, and XML markup.

Figure 1. The Semantic Web standards "stack".

The logic and reasoning can be implemented by different systems. For example, FaCT (**Fa**st **C**lassification of **T**erminologies) [30-32], Protege-2000 [49, 50], CLASSIC [45], OntoMerge [55], and DReggie [8] have been used to implement a Knowledge Base that can load and verify DAML+OIL. When fully deployed, the standard DAML+OIL XML language (or OWL) will be able to be used as a common format to load, update and query KBs implemented with different logic engines.

This experiment uses the FaCT (**Fa**st **C**lassification of **T**erminologies) [30-32] reasoning engine, implements the *SHIQ(D)* logic, a specific *Description Logic* which is expressive but can be implement efficiently [31, 32]. Figure 2 illustrates the relationship between the logics, the reasoning engine, and the XML languages. Description Logic is discussed in next section, and DAML+OIL are explained in the following section.

2.3.2. Description Logics

There are many approaches to automated reasoning; the Semantic Web has focused on *Description Logics* (also known as *Terminological Logics* or *Concept Languages*). *Description Logics* are a general class of logic are specifically designed to model vocabularies (hence the name) [11, 18-20, 23, 24, 32, 33, 35, 57, 62]. A Description Logic represents classes of individuals and roles are binary relationships used to specify properties or attributes.

Description Logics are descendants of Semantic Networks [61] and related to *frame theory* [48]. Description Logics are also related to object-oriented languages: the classes and types (but not behaviors) of an object-oriented language can be stated in a Description Logic as hierarchies of concepts and roles. When a class hierarchy is expressed in a Description Logic, the model is proved *satisfiable* if and only if the class hierarchy is correct (i.e., type checking is correct). (Of course, it is not necessary to implement a general-purpose logical system to implement type checking.)

Systems built using Description Logic are used to create a Knowledge Base, composed of two components:

- intensional: a schema defining classes, properties, and relations among classes (the *terminological knowledge*, termed the 'Tbox')
- extensional: a (partial) instantiation of the schema, containing assertions about individuals (the *assertional knowledge*, termed the 'Abox').

A Knowledge Base (KB) is a pair, (Tbox, Abox). Basically, the Tbox is the model of what *can* be true, the Abox is the model of what *currently is* true.

A *Description Logic* has a formal semantics, which can be used to automatically reason about the KB. The reasoning includes the ability to deduce answers to important questions including [19, 20, 22, 23, 32, 34, 57, 62]:

- Concept satisfiability – whether concept C can exist

- Subsumption – is concept C is a case of concept D
- Consistency – is the entire KB satisfiable
- Instance Checking – is an assertion satisfied.

These questions can represent important logical requirements for ubicomp systems.

For example, Gonzalez-Castillo [22] defines a semantic match for a query (service request) to a service (advertisement) can be implemented as logic operations on two concepts (C1, C2). C1 matches C2 if:

- C1 is equivalent to C2, or
- C1 is a sub-concept of C2, or
- C1 is a super-concept of a concept subsumed by C2, or
- C1 is a sub-concept of a direct super-concept of C2 whose intersection with C2 is satisfiable

Among formal logics, Description Logics have been demonstrated to provide substantial expressive and reasoning power with real and effective implementations.

This study uses the the FaCT reasoning engine which has a CORBA interface and implements the *SHIQ(D)* logic [4, 29]. *SHIQ(D)* logic is a specific *Description Logic* which is expressive but can be implement efficiently. The FaCT system is programmed in the OIL language [16, 17, 30]. The OIL program is compiled into a set of assertions which are used to construct a Knowledge Base (KB). The KB can be tested with FaCT to prove *satisfiability* and *subsumption*.

The *SHIQ* logic supports the concepts required for the definition of ontologies (the Tbox), but cannot express individuals (needed for the Abox). For this reason, Gonzalez-Castillo, et al. [22] argue that the *SHOQ(D)* logic should be used instead. Algorithms to implement subsumption and satisfiability are known for *SHOQ(D)* ([33, 57]), although implementations are not available at this writing.

Table 2 gives a summary the logical concepts and the DAML tags that represent them [22, 62] (for alternative statements, see also [18, 24-26]). The *SHIQ(D)* logic includes the concepts in all the rows except nominals and value restrictions (labeled “O”). These concepts are necessary to define sets of instances of concepts, and to define properties with specific values. In contrast, the *SHOQ(D)* language has all the rows of the Table 2 except inverse (labeled “I”).

The key reasoning for these logics is the determination of concept satisfiability, concept subsumption, and KB consistency. Table 3 gives a formal statement of these conditions.

Table 2. Correspondence of Description Logic and DAML (see also [19, 22, 24, 26, 60]). (Concepts A, C, D; Roles R, S; type T, D; instance o, p, d)

DL Expressiveness	DL Syntax	DAML/XMLS Syntax	Serv. Descript. Lang.
$\mathcal{ALC}$, also called $\mathcal{S}$ when transitively closed primitive roles are included	A	daml:Class	Concept
	$\top$	daml:Thing	Thing (Top)
	$\bot$	daml:Nothing	Nothing (Bottom)
	$(C \subseteq D)$	daml:subClassOf	Subsumption
	$(C \equiv D)$	daml:sameClassAs	Equivalence
	R	daml:Property	Role: Properties
	R	daml:ObjectProperty	ObjectProperties
	$(C \cap D)$	daml:intersectionOf	Conjunction
	$(C \cup D)$	daml:disjunctionOf	Disjunction
	$\neg C$	daml:complementOf	Negation
	$\forall R.C$	daml:toClass	Universal Role Rest.
	$\exists R.C$	daml:hasClass	Existential Role Rest.
$\mathcal{N}$	$\leq nR.\top$	daml:maxCardinality	
	$\leq nR.\top$	daml:minCardinality	Non-Qualifed Card.
	$= nR.\top$	daml:cardinality	
$\mathcal{Q}$	$\leq nR.C$	daml:hasClassQ	Qualifed Cardinality
		daml:minCardinalityQ	
	$\leq nR.C$	daml:hasClassQ	
		daml:maxCardinalityQ	
	$= nR.C$	daml:hasClassQ	
		daml:cardinalityQ	
$\mathcal{I}$	R^{-}	daml:inverseOf	Inverse Roles
$\mathcal{H}$	$(R \subseteq S)$	daml:subPropertyOf	Role Hierarchy: Subsumption of Roles
	$(R \equiv S)$	daml:samePropertyOf	Equivalence of Roles
$\mathcal{O}$	$\{o, p, \ldots\}$	XML Type + rdf:value	Nominals (Collection of values)
	$\exists T.\{o, p, \ldots\}$	daml:hasValue	Value Restrictions
$(\mathcal{D})$	D	daml:Datatype + XMLS	Datatype System
	T	daml:datatypeProperty	Datatype Property
	$\exists T.d$	daml:hasClass + XMLS Type	Exist. Datat. Rest.
	$\forall T.d$	daml:toClass + XMLS Type	Univ. Datat. Rest.

Table 3. Definition of logical queries for a description logic (e.g., see also [19, 24, 60]). Concepts C, D, instance a.

Query	For a KB, Σ = < Tbox, Abox >
Concept satisfiability	$\Sigma \ \models\!\!\!\!/\ C \equiv \bot$
Concept subsumption	$\Sigma \models C \subseteq D$ (i.e., $\Sigma \ \models\!\!\!\!/\ C$ int $\neg D \equiv \bot$)
Consistency	$\Sigma \ \models\!\!\!\!/\ \bot$
Instance satisfiability	$\Sigma \models C(a)$

One of the important reasons for using a description logic is that the satisfiability and subsumption can be computed efficiently. These questions are undecidable for general First Order Logic, and many otherwise desirable logics are also computationally intractable. Efficient algorithms for *SHIQ* are given in [34], and implemented in the FaCT server [4, 31]. Similar algorithms are proven for *SHOQ* [33, 57] , and implementations should be forthcoming.

Briefly, these algorithms convert the assertions to a normal form, and construct a (large) set of constraints, which are then analyzed to prove a contradiction or not. The constraints are propagated through the graph, which must be pruned by careful heuristics. These algorithms can provide good performance for typical cases, though the problems are intractable in the worst case. See [34] and [57] for more detailed explanation of efficient algorithms.

The Tbox (the terminological knowledge) is the intensional knowledge. The Tbox statements are typically definitions such as: C, A $\subseteq$ C, and so on. The Tbox assertions can be reasoned on used to prove the KB is consistent, concepts are subsumed, as in the first three rows of Table 3. These proofs are used to implement automatic classification and matching [15, 62, 67].

The Abox (the assertional knowledge) is the extensional knowledge, statements about individuals and properties. The assertions are typically C(a) (*a is-a C*), and R(a,b) (*a is related to b by role R*). The assertions can be checked for *instance satisfiability* (as in Table 3). This proof implies that the instance does not conflict with the schema or any other assertions currently in the KB.

2.3.3. Semantic Web Software

This experiment is made possible by the use of available free software with open interfaces. The FaCT reasoning engine is a stand-alone server with a CORBA interface [4, 29, 31]. The interface is essentially the OIL language, plus logic queries (*satisfiability* and *subsumption*). The OIL program is compiled into a set of assertions which are sent to the FaCT server to construct a Knowledge Base (KB). The KB can be tested with FaCT to prove *satisfiability* (logical consistency) and *subsumption* (logical equivalence).

The *uk.ac.man.cs.img.oil* package is available as part of the OILed tool [54]. This package implements reading and writing DAML+OIL XML documents. A DAML document is translated into an internal data structure (*Ontology*). The *oil* package can verify the ontology by converting it to a series of assertions in OIL, which are sent to the FaCT reasoner to create a Knowledge Base (KB). The *oil* package then queries to test that the classes and individuals (instances) in the ontology are *satisfiable* in the KB. If every class and instance in the FaCT KB is *satisfiable*, then the KB is consistent and the ontology is correct.

Figure 3 shows the main components used. Together, these packages are capable of validating any OIL ontology from a DAML XML file. In addition, the OILed tool [54] can be used to create and validate DAML files. Furthermore, ontologies can import other ontologies (using XML namespaces), and the *oil* package can create and validate an ontology composed from multiple DAML files retrieved from the Internet.

2.4. Ontologies

The terminology or vocabularies used by a domain is developed to express the concepts that the experts in this domain need to exchange information on the topic. The terms represent the essential concepts of the domain. However, the specific terms used are, of course, arbitrary. This leads to the classic problems of vocabulary control in information systems [38]: in many cases, the same concept or very similar concept may have many different terms applied to it in different domain contexts. Humans are quite used to quickly switching and matching words from different contexts. Indeed, specialized technical training involves learning domain vocabularies and mapping them to other

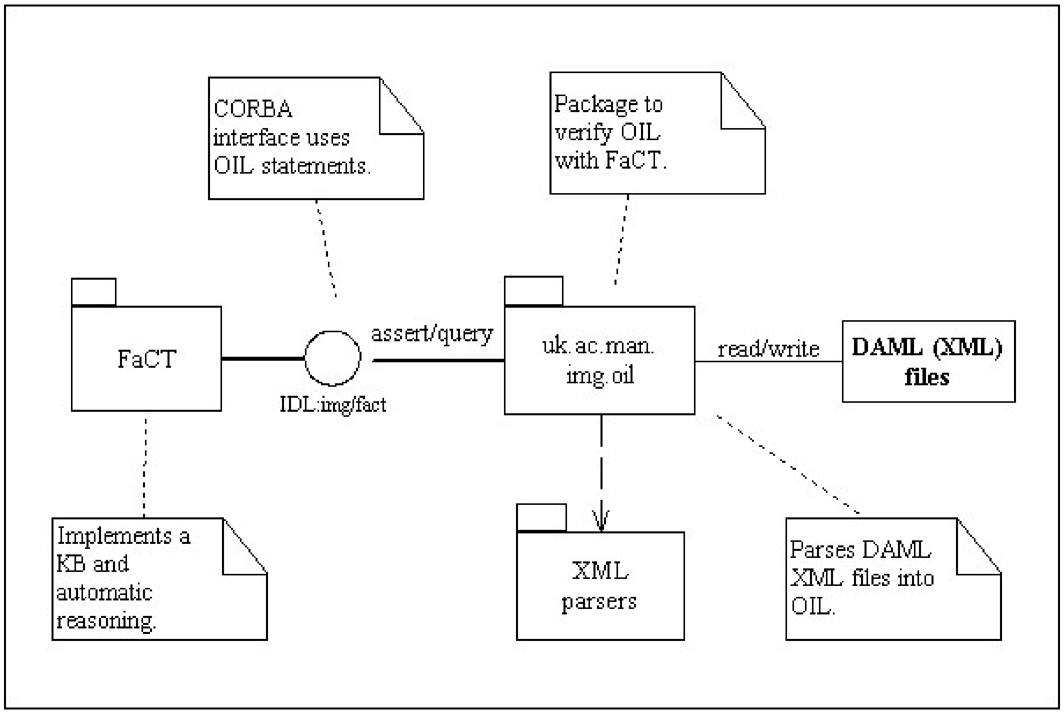

CORBA interface uses OIL statements.
Package to verify OIL with FaCT.
FaCT
assert/query
IDL:img/fact
uk.ac.man. img.oil
read/write
DAML (XML) files
Implements a KB and automatic reasoning.
XML parsers
Parses DAML XML files into OIL.

Figure 3. The logic programming components.

domain vocabularies. Unfortunately, this process is very difficult for machines [65].

Domain experts and standards bodies will define the concepts develop the formal vocabulary for domains reusing higher-level vocabularies and vocabularies from other domains when they are available and apply. An important goal of an *ontology* is to formalize this process, and to generate a formal specification of the domain-specific vocabulary.

An *ontology* is a formal vocabulary and grammar of concepts [14, 23, 81]. The Semantic Web XML languages addresses this process with schemas based on formal *ontologies.* The Ontology Information Language (OIL) language is an XML-based language that enables such information to be retrieved in an open network [10, 17, 66]. The OIL is not simply a record format, it defines logical rules to enable the document to be validated (proved correct) and then interpreted into a specific local schema.

Using the DARPA Agent Markup Language (DAML), a query can refer to the ontology used to construct it, with a URL for an OIL document [1, 9]. In turn, the receiver can retrieve the ontology if needed, parse it, and interpret the query into its own preferred vocabulary. Similarly, the OIL can be used to publish the schema (ontology) of the library as an XML document. This mechanism enables the parties to share their schemas at run time, using a standard machine interpretable format.

The next section explains how ontologies are used in out Pervasive Computing Environment.

3. Kinds of Ontologies in GAIA

We use ontologies to describe various parts of our pervasive environment, GAIA. In particular, we have ontologies that have meta-data about the different kinds of entities in our environment. We also have ontologies to describe the different kinds of contextual information in our environment. In future work, we will investigate other uses of ontologies, including generic descriptions of tasks and policies.

3.1. Ontologies for different entities

Pervasive computing environments have a large number of different types of entities. There are different kinds of devices ranging from small wearable devices and handhelds to large wall displays and powerful servers. There are many services that help in the functioning of the environment. These services include Lookup Services, Authentication and Access Control services, Event Services, etc. There are different kinds of applications like music players, PowerPoint viewers, drawing applications, etc. Finally, there are the users of the environment who have different roles (like student, administrator, etc.). Ontologies provides a standard, machine-readable, taxonomy of the different kinds of entities. We have developed ontologies that define the different kinds of entities, provide meta-data about them and describe how they relate to each other. These ontologies are written in DAML+OIL.

Table 4, Some of the classes in the ontology

Entity	Class of all objects in the system - includes all applications, services, devices and users
Service	Subclass of "Entity", the Service Class encompasses all those components that provide some form of service (!!) It includes both kernel services like Space Repository, etc. as well as other services like Context Providers.
CommandableService	A subclass of "Service", it includes all those services to which you can send a command to be executed
SearchableService	A subclass of "Service", it includes all those services to which you can send a query and then get a set of results in return
MP3Server	A subclass of both CommandableService and SearchableService, it maintains a list of songs - this list can be searched by certain attributes and it can also be sent commands to play songs
Application	Subclass of Entity, this represents the class of all applications in the environment - eg. powerpoint, scribble applications, etc.
PowerPointApplication	Subclass of Application, this class describes the different kinds of PowerPoint Applications
User	Subclass of entitiy, this is the class of all users (or people) in the environment
Device	Subclass of entitiy, this is the class of all devices in the system - UOBHosts, cameras, fingerprint recognizers, etc.

In addition to ontologies that provide meta-data about the different classes of entities. each instance (or individual) also has a description in DAML+OIL that gives the properties of this instance. This DAML+OIL description must be consistent with the meta-data description of the class in the ontology. For example, the ontology has a class called MP3File and it requires all instances of this class to have certain attributes like artist, genre, album, length, etc.. Thus, every description of an MP3 file has to have these fields. The description of every instance is checked to see that it is satisfiable with the concepts defined in the ontology.

Some of the classes in our ontology that describe entities (along with a brief description of them) are shown in Table 4. Figure 4 shows the logical hierarchy of these classes.

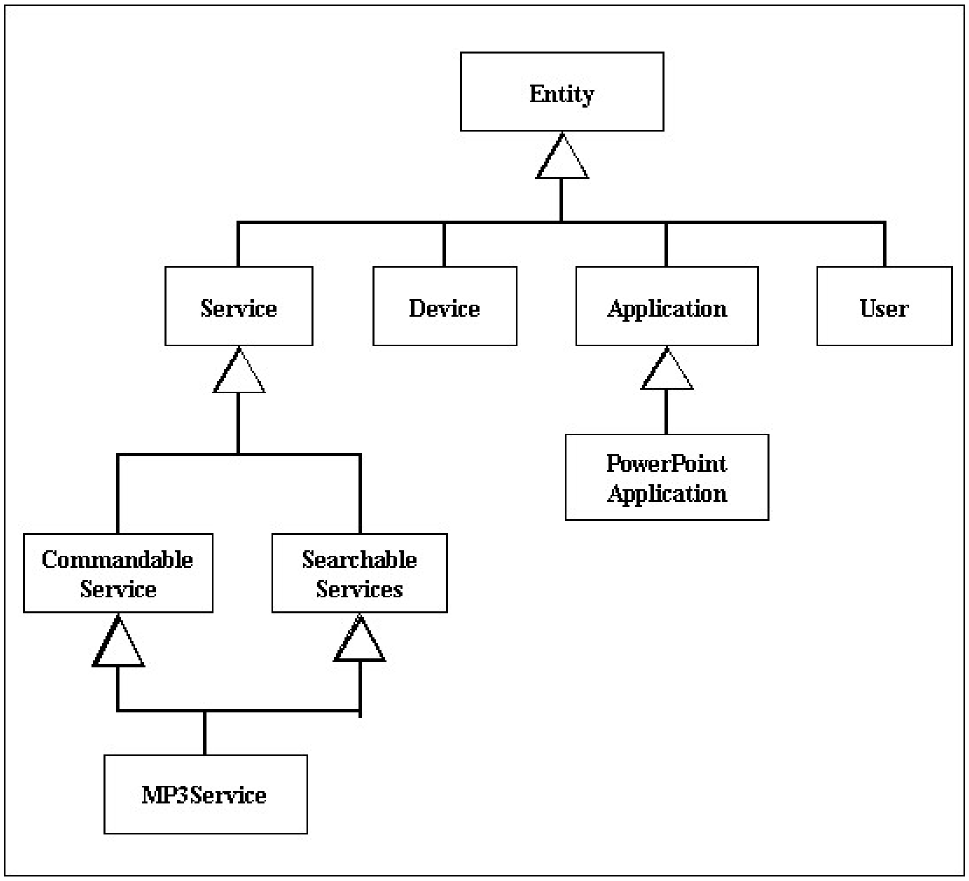


Figure 4. The logical hierarchy of classes from Error! Reference source not found.**.**

A Pervasive Computing Environment is very dynamic. New kinds of entities can be added to the environment at any time. The Ontology Server allows adding new classes and properties to the existing ontologies at any time. For this, a new ontology describing the new entities is first developed. This new ontology is then added to the shared ontology using bridge concepts that relate classes and properties in the new ontology to existing classes and properties in the shared ontology. These bridge concepts are typically in subsumption relations that define the new entity to be a subclass of an existing class of entities. For example, if a new kind of fingerprint recognizer is added to the system, the bridge concept may state that it is a subclass of "AuthenticationDevices".

3.1.1. An example of a class in our ontology

Each type of entity in GAIA is described a class in our ontology. This class defines all properties of the entity like the search interfaces it exposes, the types of commands that can be sent to it, the data-types it deals with, etc.

As an example, we have included a part of the description of an MP3 Server in Listing 1, below. This entity maintains a set of songs in MP3 format in its database. It allows other entities to search this set of songs using various parameters like name of artist, type of song, etc. It can also be sent commands for playing songs – other entities can either request a particular song to be played or a random song to be played. In addition, there is a human-understandable description about the entity. This is specifically meant for the average user who wants to know more about the entity in a simple language.

The entity is described in terms of restrictions on various properties. The superclasses of an entity also give more of an idea about the entity. In the case of the MP3 Server, it is declared as a subclass of SearchableService (Listing 1, lines 12-16) and of CommandableService (lines 17-21) – this means it supports searches and execution of commands. Other properties of the MP3Server according to its description are that it executes MP3Files (lines 22-33), it's search schema is defined in the class MP3Attributes (lines 34-45), and that there are two types of commands that can be sent to it – MP3ServerPlay (lines 46-57) and MP3ServerRandomPlay (lines 58-69). In addition, there is a human-understable description of the class (lines 5-8).

The DAML XML maps to statements of Description Logic (e.g., see [18, 25, 26, 62]), which can be asserted to a Knowledge Base and logically validated.

3.2. Ontologies for context information

GAIA has a context infrastructure that enables applications obtain and use different kinds of contexts. This infrastructure consists of sensors that sense various contexts, reasoners that infer new context information from sensed data and applications that make use of context to adapt the way they behave. We use ontologies to describe context information. This ensures that the different entities that use context have the same semantic understanding of contextual information.

The use of ontology to describe context information is useful for checking the validity of context information. It also makes it easier to specify the behavior of context-aware applications since we know the types of contexts that are available and their structure.

There are different types of contexts that can be used by applications. These include physical contexts (like location, time), environmental contexts (weather, light and sound levels), informational contexts (stock quotes, sports scores), personal contexts (health, mood, schedule, activity), social contexts (group activity, social relationships, whom one is in a room with), application contexts (email, websites visited) and system contexts (network traffic, status of printers).

We represent contexts as predicates. We follow a convention where the name of the predicate is the type of context that is being described (like location, temperature or time).

The structure of the context predicate depends on the type of context. This structure is defined in the ontology. For example, location context information must have three fields - a subject that is a person or object, a preposition or a verb like "entering," "leaving," or "in" and a location like a room or a city. For instance, *Location (Chris , entering , room*

3231) is a valid location context. Each type of context corresponds to a class in the ontology. The fields of the context are defined as restrictions on this class.

Other example context predicates are:

- *Temperature (room 3231 , "=" , 98 F)*
- *Sister(venus , serena)*
- *StockQuote(msft , ">" , $60)*
- *PrinterStatus(srgalw1 printer queue , is , empty)*
- *Time(New York , "<" , 12:00 01/01/01)*

3.2.1. An example of using ontologies to describe context

Each type of context is defined by a class in the ontology. As an example, we give the DAML+OIL description of temperature context in Listing 2, below. According to this description, the "Temperature" context is a subclass of the more generic "WeatherInformation" context (Listing 2, lines 10-15). Other information about this context is that it consists of a subject, which can be either a "PhysicalPlace" or a "Person" (lines 16-50); it has a relater which is a "ComparisonOperator" (lines 51-64); and it has an object which is of type "TemperatureValue" (i.e. either in Centigrade or in Fahrenheit) (lines 65-78). An instance of a temperature context based on this description is *Temperature (Champaign, ">" , 40F).*

4. Use of Ontologies in GAIA

The Semantic Web technologies (Ontologies in DAML+OIL XML, a Knowledge Base, wrapped with a standard Ontology Server interface) are applied to the problems discussed above.

The ontologies that describe entities and context information are used to enable different parts of the pervasive environment interact with each other easily. In this section, we describe some of the ways in which ontologies are used in our pervasive environment, GAIA.

The Ontology Server can be used by any application, component, or service in the GAIA environment. For example, the CORBA Trading Service [37, 53] was augmented to use the Ontology Server in three ways:

- to generate CORBA Service Types (schemas)
- to create templates for CORBA Service Offers (advertisements)
- to check proposed Service Types and Service Offers against the ontologies (validation)

In this use, the ontologies provide a formal schema definition language for the CORBA Trading Service.

Other entities in the environment can query the Ontology Server to get descriptions and properties of classes. The Ontology Explorer supports queries like getting properties of other entities, definitions of terms, descriptions of different types of contextual information.

4.1. Applications of Ontologies and Semantic Services

The ontologies that describe entities and context information are used to enable different parts of the pervasive environment interact with each other easily. In this section, we describe some of the ways in which ontologies are used in our pervasive environment, GAIA.

4.1.1. Defining terms used in the environment

One of main uses of ontologies in a Pervasive Computing Environment is that it allows us to define all the terms that can be used in the environment. Ontologies allow us to attach precise semantics to various terms and clearly define the relationships between different terms. It, thus, prevents semantic ambiguities where different entities in the environment have different ideas of what a particular term means. Different entities in the environment can refer to the ontology to get a definition of a term, in case they are not sure.

For example, we have defined the term “meeting” as a subclass of “GroupActivity”. A meeting is defined to have a location, a time, an agenda (optional) and a set of participants. It has a human-understandable comment that goes as follows
“A meeting is an activity that is performed by a group of people. A meeting involves different people coming together at a particular time or place with a common purpose in mind”. Thus, both humans and automated entities in the environment can get a clear understanding of the term “meeting” by looking it up in the ontology.

4.1.2. Configuration Management: Validating Descriptions

A key advantage of using ontologies for describing entities and contextual information is that we can determine whether these descriptions are valid with respect to the schema defined by the ontology. When a new entity is introduced into the system, its description can be checked against the existing ontology to see whether it is satisfiable. If the description is not consistent with the concepts described in the ontology, then either the description is faulty (in which case the owner of the entity/context has to develop a correct description of the entity/context), or there are safety or security issues with the new entity or context. For example, the ontology may dictate that the power of a bulb in the environment should have a value between 20 and 50 Watt. In that case, if somebody tries to install a new 100 Watt bulb, then the description of the new bulb would be inconsistent with the ontology and a safety warning may be generated.

When a new entity is first introduced into the environment, it is described in DAML+OIL, which is sent to the Ontology Server to assure that the description of this instance is logically consistent with the definition of the class of the entity and all the logical relations of the current Knowledge Base. If there is a logical inconsistency, then the developer of that entity is required to revise the description of the entity (or change the properties of the entity) to ensure that it does meet the constraints defined in the ontologies. The operation of checking the logical consistency of the description of an entity is computationally intensive; and hence is performed only the first time the entity is introduced into the environment (or whenever the description of the entity changes). It is not performed every time the Space is bootstrapped.

Formal ontologies increase the ability to use descriptions from different, autonomous sources. The DAML+OIL ontologies can be published, to enable autonomous developers and service providers to describe their products with the correct vocabulary. Conversely, autonomous entities can specify the correct formal vocabulary to be used to interpret their descriptions by referring to the relevant DAML+OIL ontology. These actions require more than the URL: the formal semantics defined for DAML+OIL ensures that ontologies from different sources can be used together.

4.1.3. Semantic Discovery and Matchmaking

A Pervasive Computing Environment is an open system, in which the components are heterogeneous and autonomous. Before entities can compose and collaborate to deliver services, they must *discover* each other. Conventional object registries provide a limited capability for object discovery, and so-called *discovery protocols* (such as Salutation [58] or JINI [13]), support limited ability to spontaneously discover entities on a network. For a pervasive system, these protocols must be enhanced to provide semantic discovery [41]: it must be possible to discover all and only the “relevant” entities, without knowing in advance what will be relevant. This process has also been termed “matchmaking” [73].

Semantic discovery can involve several related activities: advertising, querying, and browsing. In each case, the parties exchange structured records describing the offered service (advertising, response to query) or the desired service (querying). The exchange may be manual (browsing), real-time (a query to discover the current local state of the system), persistent (a standing query, i.e., to be notified). The exchange may be a push (advertisement, notification), pull (query), or some combination. In all cases, it is critical that the data is filtered, to select a set that best matches the intentions of the parties. [73] summarizes these requirements.

Object registries, such as the CORBA Naming Service [51], provide a basic mechanism for finding well-known (i.e., known in advance) services. Brokers, such as the CORBA Trader Service [53], provide the capability to locate services by attributes. Many other services provide similar features, including LDAP [96], JINI [13], and Microsoft’s Registry [56].

In the case of a Pervasive Computing Environment, the entities of interest are the active components of the system, which includes devices, services, and physical entities in the environments. We define ontologies for describing different categories of entities, and use the Semantic Web technologies to enable semantic discovery and matchmaking across the many kinds of entities.

One of the main issues with traditional discovery services is that in a massively distributed environment with a large number of autonomous entities, it is unrealistic to expect advertisements and requests to be equivalent, or even that there exists a service that fulfills exactly the needs of the requester. Advertisers and requesters could have very different perspectives and knowledge about the same service. Semantic discovery aims to bridge this semantic gap between advertisers and requesters. A service that tries to provide semantic discovery would use its knowledge of the environment and its semantic understanding of the advertisement and the request to recognize that the two are related, even if they, say, use different terms or different concepts.

DAML+OIL is based on description logics, that supports some of the operations required for semantic discovery like classification and subsumption. DAML+OIL also allows the definition of relations between concepts.

Variations of discovery and matchmaking are required for many functions of a ubiquitous computing environment. This section discusses three different kinds of discovery: human interaction, searches, and interaction of components.

4.1.4. Specifying Rules for Context-Sensitive Behavior

A key feature of applications in pervasive computing environments is that they are context-aware, i.e. they are able to obtain the current context and adapt their behavior to different situations. For example, a music player application in a smart room may automatically play a different song depending on who is in the room and it may decide the volume of the song depending on the time of day. GAIA allows application developers to specify different behaviors of their applications for different contexts. We use ontologies to make it easier for developers to specify context-sensitive behavior.

Context-aware applications in GAIA have rules that describe what actions should be taken in different contexts. An example of a rule is :

> *IF Location(Roy, Entering, Room 2401) AND Time(morning) THEN play a rock song.*

A rule consists of a condition, which if satisfied, leads to a certain action being performed. The condition is a Boolean expression consisting of predicates based on context information.

In order to write such a rule, an application developer must know the different kinds of contexts available as well as possible actions that can be taken by the application. We have ontologies that describe the different kinds of context information – location, time, temperature, activities of people, etc.. We also have ontologies that describe different applications and what commands can be sent to them. The ontologies greatly simplify the task of writing rules. We have a GUI which allows developers to write rules easily. The GUI allows him to construct conditions out of the various possible types of contexts available. It then allows him to choose the action to be performed at these contexts from the list of possible commands that can be sent to this application as described in the ontology. Developers can, thus, very quickly, impart context-sensitivity to applications.

4.2. Example Uses of Ontologies and Semantic Services

The use of ontologies can improve several aspects of the Pervasive Computing System. In general, communication between autonomous entities is improved by the use of ontologies. This section discusses several examples of such communications: human interaction, search, and interaction of components.

4.2.1. Better Interaction with Humans

An important part of pervasive computing environments are the humans in the environment. These environments automate several tasks and proactively perform various actions to make life easier for the humans. Ontologies can be used to make better user interfaces and allow these environments to interact with humans in a more intelligent way. Very often users, especially novice users, do not know what various terms used in interfaces mean or how different parts of the system are related to each other. The problem is especially acute in pervasive environments with its myriad devices, applications and services. It is very easy for users to get lost in these environments especially if they do not have a clear model of how the system works. Ontologies can be used to alleviate this problem. Ontologies describe different parts of the system, the various terms used and how various parts interact with each other. All classes and properties in the ontology also have documentation that describe them in greater detail in user-understandable language. Users can thus browse or search the ontology to better understand the system. Ontologies enable semantic interoperability between users and the system.

We have developed a GUI called the Ontology Explorer that allows users to browse the ontology describing the environment. Users can search for different classes in the ontology. He can then browse the results–for example, he can get documentation about the classes returned, get properties of the class, etc.. He can also get instances of the class. For example, if the user searches using the string “MP3”, he gets all classes in the Ontology that deals with “MP3” – this includes an MP3 Server, MP3 Files, MP3 Attributes, etc. He can then get more details about the classes. He can get instances of MP3 Files and interact with the MP3 Server, as described in the next sections. More details about the Ontology Explorer as well as screenshots can be found in the Implementation section.

4.2.2. Improved Searches

One of the most frequent activities in computing is search. Both users as well as computer programs need to search data sources for relevant information. Components that allow searches to take place expose their schemas in the ontology. They can also specify which fields in the query are required and which are optional. Thus any entity can browse the ontology to learn the schema and query formats supported by the searchable component. They can then frame their query and get the results. We also generate search interfaces based on the schema which humans can use to enter queries. This greatly speeds development time, since each component that allows searches need not have a separate GUI for users. Instead, all they have to do is to specify their schema in an ontology – the schema is then used to automatically generate the interface.

These ontology-driven user interfaces makes query formulation easier. The user can't make a mistake by, say, using unknown terms. All available attributes and fillers are automatically loaded and presented dynamically depending on the query-template specified in the ontology. The user frames his query by just choosing reasonable values for the given attributes.

For example, the MP3 Server supports searches based on attributes like name of song, genre of song, length of song, etc. This schema is described in the Ontology. Other agent can, thus, get the schema from the Ontology Server and send queries to the MP3 Server. Users can also send queries to the MP3 Server using the Ontology Explorer. The Ontology Explorer gets the schema from the Ontology Server and generates a dialog (based on the schema) where the user can enter the query. For example, the user can search for all songs by Elvis Presley. The Ontology Explorer submits the query to the MP3 Server and displays the results for the user. More details about how the Ontology Explorer is used to let users perform searches as well as screenshots can be found in the Implementation section.

Similarly, automated agents can also make use of the search schemas defined in the ontology to frame queries to other entities and get the results. This smoothens the interactions between different entities.

A more difficult problem is to provide context-sensitive queries and responses: the user frames the request in the vocabulary of his application task and context, but this may not match the vocabulary of the system. It will be necessary to translate requests to equivalent vocabularies, and to translate responses to the vocabulary of the consumer. In general, such translations are very difficult and cannot be done automatically. But when translations are known (e.g., between two standard vocabularies), ontologies can be used to automatically transform queries and responses.

4.2.3. Allowing Easier Interaction With Components

Search is just one of the activities that users and computer programs can perform on various components in a pervasive environment. Different components allow different types of actions to be performed on them. For example, a music player allows different commands to be send to it –start, stop, pause, change volume, etc.. In our framework, components specify the commands they support and the parameters of these commands in an ontology. Thus, other entities can learn what commands can be sent to a particular component and can thus easily interact with this component. As in the case of search, we can easily generate GUIs where users can specify commands to be sent to a particular component.

The ontology, thus, provides a generic way of interacting with different agents. The ontology describes the different commands that can be sent to an agent. For each command, it also describes what arguments or parameters are needed. Other agents, as well as users, can thus send these commands with the correct parameters to the agent.

The Ontology Explorer also allows users to send commands to different agents. For example, the MP3 Server supports commands like play, stop, pause, increase volume, etc. If the user wants to send a command to this MP3 Server, the Ontology Explorer opens up a dialog that lists the commands available. Once the user chooses a command, it gets the list of required parameters for the command from the Ontology Server and allows the user to fill in these parameters. For example, if the user chooses the "play" command, the Ontology Explorer discovers that the play command needs one parameter – the name of the song. It then presents the user with a list of songs (obtained from the MP3 Server) and allows the user to either choose a song or enter the location of a new song. It then sends the play command to the MP3 Server. More details about how the Ontology Explorer is used to let users send commands as well as screenshots can be found in the next section.

Similarly, automated agents can also make use of the commands defined in the ontology to send commands to other entities. This smoothens the interactions between different entities.

5. Implementation Details

We have integrated the use of ontologies in our smart spaces framework, GAIA. All the ontologies in GAIA are maintained by an Ontology Server. Other entities in GAIA contact the Ontology Server to get descriptions of entities in the environment, meta-information about context or definitions of various terms used in GAIA. It is also possible to support semantic queries (for instance, classification of individuals or subsumption of concepts). Such semantic queries require the use of a reasoning engine that uses description logics like the FaCT reasoning engine. We plan to provide support for such queries in the near future.

One of the key benefits in using ontologies is that it aids interaction between users and the environment. With that aim in mind, we have developed an Ontology Explorer which allows users to browse and search the ontologies in the space. The Ontology Explorer also allows users to interact with other entities in the space through it. The interaction with other entities is governed by their properties as defined in the ontology.

5.1. The Ontology Server

The Ontology Server is a CORBA service maintains a single, cumulative "current ontology" for an Active Space. Each Active Space has one Ontology Server running in it.. As described above, the ontology is a logical schema for all the entities of the system. The Ontology Server implements algorithms to load and validate ontologies from DAML+OIL XML files, compose ontologies into a combined system ontology, and serve

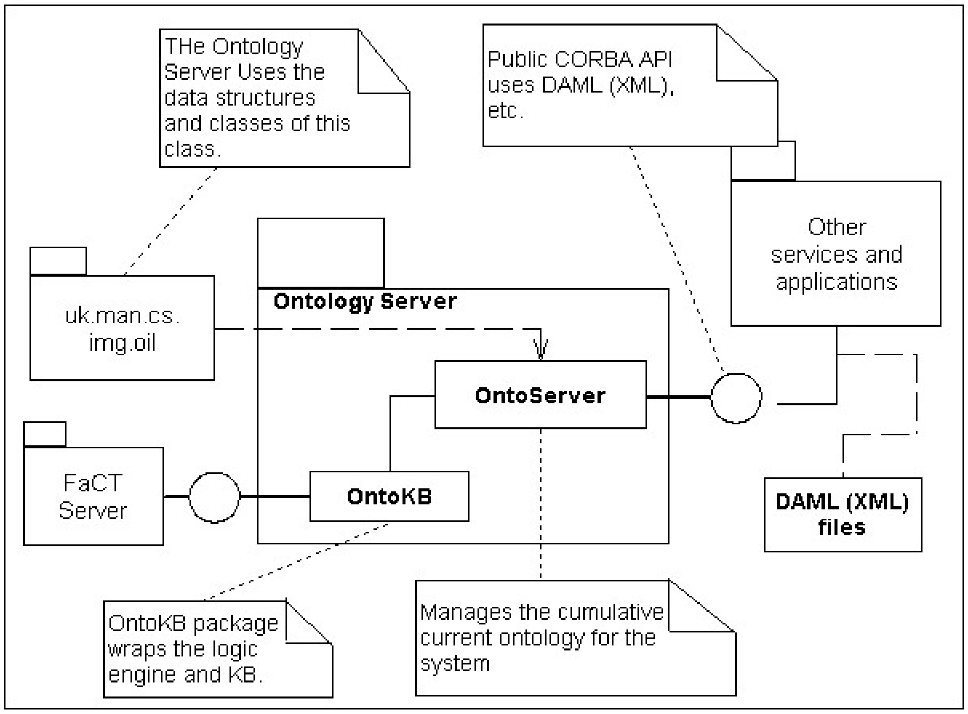

THe Ontology Server Uses the data structures and classes of this class.
Public CORBA API uses DAML (XML), etc.
Other services and applications
Ontology Server
uk.man.cs. img.oil
OntoServer
FaCT Server
OntoKB
DAML (XML) files
OntoKB package wraps the logic engine and KB.
Manages the cumulative current ontology for the system

Figure 5. Overview of the OntologyService.

logical queries to a Knowledge Base (KB) representing the dynamically composed ontology [42].

5.1.1. Architecture

Figure 5 shows the key components of the Ontology Server. The service has a CORBA interface, and two main components:

- The OntolServer, which implements the interface, maintains the current ontology and other state information, and executes the algorithms defined in the previous section.
- The OntoKB, a private class which is a generic wrapper for the logic engine and KB.

The Ontology Server interface uses DAML+OIL XML documents to define ontologies and individual objects (as well-formed fragments of ontologies). The Ontology Server interface uses only open, public objects and formats, hiding the details of the data structures, logic engine, and KB. This makes it possible to substitute alternative implementations of the ontology data structures, logic engine, and KB.

The OntoKB class implements a generic interface for a Knowledge Base, including load, update, validate, and query.

5.1.2. Content and Behavior of the Knowledge Base (KB)

In this implementation, the KB managed by the Ontology Server only has class information: the types or classes of different entities or terms, not descriptions of actual instances of entities (i.e., the current state of the system). This class information is sufficient for carrying out most of the tasks we are interested in (which will be described in the following sections).

A KB of description of instances would be far more dynamic than description of classes. Since instances can enter and leave the environment at any time, the Knowledge Base may have to be continuously updated to keep it synchronized with the current state of the space. Also, there potentially may be a very large number of instances of entities (compared to the number of classes). It would be very challenging to implement a KB that could efficiently handle large numbers of updates in real time. A KB of instances also requires a naming scheme so instances can be reliably recognized and distinguished, and would need robust error handling and recovery. The KB of class information is smaller and less volatile, so it could be implemented.

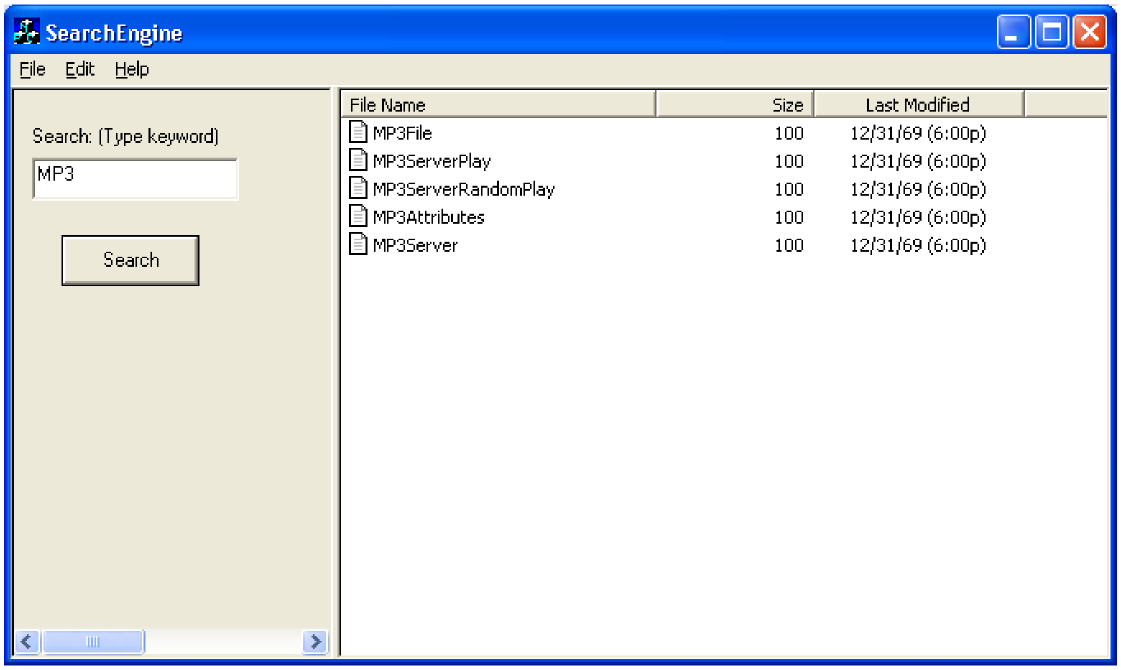
The information about existing entities is managed by other components of GAIA. GAIA has a service called the Space Repository which maintains information about the entities in the space at any time. Each entity has an XML description which is written in accordance to the meta-information about the entity as described in the ontology. The Space Repository maintains the descriptions of all entities that are currently in the space. More details about the Space Repository can be found in [63]. Instances of context information are distributed among different sensors and other entities that use context. The Ontology Server manages a unified ontology so that these distributed services can interoperate.

5.2. Integration into GAIA Framework

The Ontology Server has been integrated into the GAIA framework, to create a prototype semantic infrastructure. Figure 6 shows the interaction of the Ontology Server, GAIA entities, and the Ontology Browser.

The Ontology Server has access to the ontologies described in Section 3. These ontologies are loaded into the Ontology Server when it is started. The Ontology Server also asserts the concepts described in the ontologies in the FaCT Reasoning Engine to make sure that they are logically consistent. It registers with the CORBA Naming Service so that it can be discovered by other entities in the environment.

Other entities in the environment can query the Ontology Server to get descriptions and properties of classes. The Ontology Explorer supports queries like getting properties of other entities, definitions of terms, descriptions of different types of contextual information. Since the Ontology Server is a CORBA Object, it is easy for other CORBA-Based entities to get a reference to it from the CORBA Naming Service and then interact with it.



File Name	Size	Last Modified
MP3File	100	12/31/69 (6:00p)
MP3ServerPlay	100	12/31/69 (6:00p)
MP3ServerRandomPlay	100	12/31/69 (6:00p)
MP3Attributes	100	12/31/69 (6:00p)
MP3Server	100	12/31/69 (6:00p)

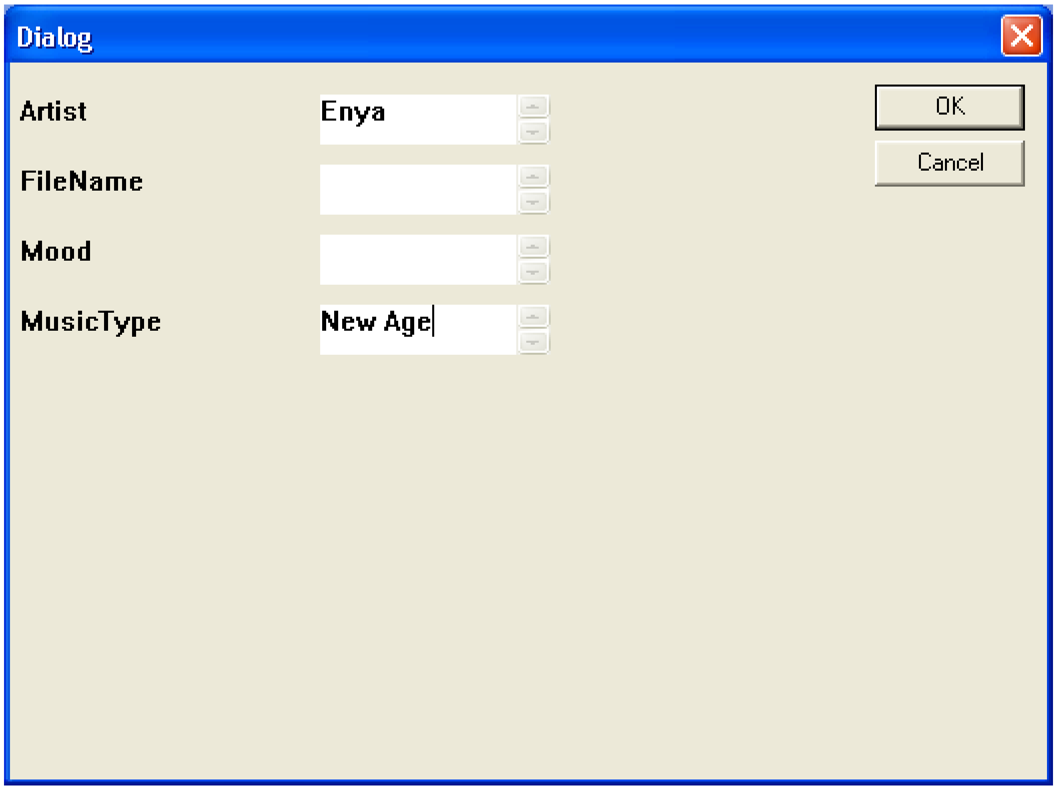
Figure 7. The results of a search for "MP3".

5.3. The Ontology Explorer

One of the key benefits in using ontologies is that it aids interaction between users and the environment. With that aim in mind, we have developed an Ontology Explorer which allows users to browse and search the ontologies in the space. The Ontology Explorer also allows users to interact with other entities in the space through it. The interaction with other entities is governed by their properties as defined in the ontology. This Ontology Explorer is similar to a class browser, except it has information about all the entities of the system, not just the software classes.

The Ontology Explorer GUI allows searching the ontology and interacting with different entities in the environment with the help of the ontology. It can perform a keyword-based search on all the classes and properties in the ontology. The user can then browse the results returned – for example, he can get documentation about the classes returned, get properties of the class, etc. He can also get instances of the class. This is done by contacting a repository that maintains information about the instances of the class of entities. Figure 7 shows the result of a query on the keyword "MP3". The result is a set of classes that are related to MP3 files and services.

If the class supports searches (for example, if they are databases), the user can enter queries that are sent to an instance and the results are then displayed. To support such searches, the Ontology Search Engine gets the schema for searching the instance from the Ontology Server and generates a GUI where the user can enter values for the query. For example, Figure 8 shows the query form for searching the MP3 Service for MP3 files.



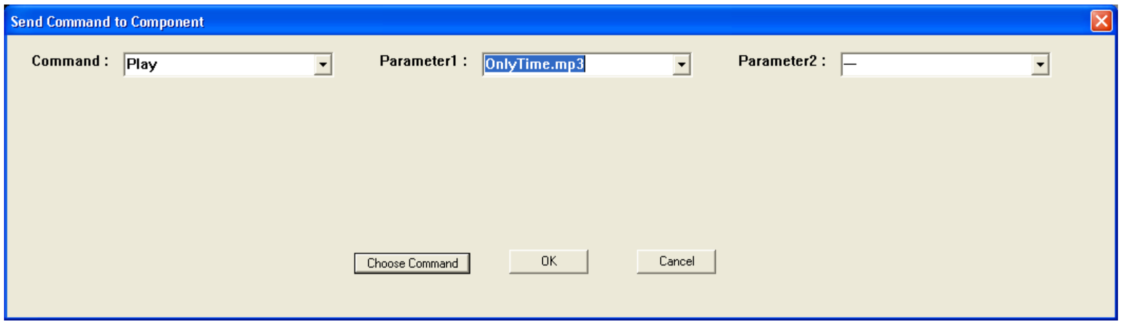
Artist	Enya
FileName	
Mood	
MusicType	New Age

Figure 8. The search dialog for MP3 files (automatically generated from ontology information)

The attributes of the MP3 that may be searched (Artist, MusicType, etc.) were automatically retrieved from the ontology. If new or different attributes are added in the future, the ontology will be changed and the Explorer will automatically pick up the new fields.

Some entities support commands being sent to them. The Explorer gets the type of commands that an entity supports as well the parameters for these commands from the Ontology Server. It then displays a GUI where the user can frame his command and send it for execution to the entity. For example, the MP3 Server supports various commands like Play, Pause, Stop, etc.

Figure 9 shows an example of how a command can be sent to the MP3Server. The user can choose the command he wants to send from a list of available commands. Once he chooses the command (say “Play”), the Ontology Explorer queries the Ontology Server to see if this command requires any parameters, and if it does what the of values should those parameters be. In the example below, the “Play” command has been defined to require one parameter–the name of the song. The Ontology Explorer asks the MP3 Server for a list of songs in its database; it then displays the list of songs to the user and the user can choose the song he wants to play.



Command : Play
Parameter1 : OnlyTime.mp3
Parameter2 : --

Figure 9. The command dialog for an MP3 Player (automatically generated from ontology information.)

The GUI was developed using C++, and it uses CORBA to communicate with other entities in GAIA.

6. Discussion and Future Work

This study has integrated semantic web technology into the GAIA infrastructure. This study has shown the need for future work in several areas. Some of these issues are briefly discussed here.

6.1. Important Findings

This study has shown that the Semantic Web technology can be used with CORBA-based infrastructure to solve some problems for a Pervasive Computing Environment. The Ontology Server provides a standard interface to a Knowledge Base and logic engine. Ontologies for descriptions of entities and relationships are developed with knowledge engineering environment and written as DAML+OIL XML files. Components of the system use the CORBA-based infrastructure to update and query the Ontology Server.

The DAML+OIL (and in the future, OWL) languages exploit the advantages of XML, and add a standard logical model, to make each DAML+OIL XML document a logic program. The logical model allows DAML+OIL to be loaded into a Knowledge Base, which can apply automated reasoning. This study integrated DAML+OIL and an example Knowledge Base into our CORBA-based Pervasive Computing Environment.

Conventional distributed systems and the emerging Web Services architecture require standards for defining, managing, and exchanging schemas. The Semantic Web technology can be used to solve some of these problems. Furthermore, our study shows that the same technology can be used to solve critical problems for Pervasive Computing.

The DAML+OIL language adds the advantages of the XML standard: a universally parseable representation, a universal standard for namespaces, widely available software support across many platforms, and so on. These features are especially important for implementing multiple vocabularies (schemas) from autonomous sources: XML provides the critical interoperability that enables the publication and exchange of vocabularies.

Again, the DAML+OIL language uses the mechanisms of XML to deliver *well-defined logic programs*.

6.1.1. Limitations of Description Logic

The DAML and the Description Logic (DL) underlying DAML are necessary but not sufficient for ubiquitous computing applications. Specifically, Description Logics are not suited for some critical aspects of ubiquitous computing: DL does not deal well with quantitative concepts; including order, quantity, time, or rates. Unfortunately, this kind of reasoning is essential to certain aspects of ubiquitous computing, including, for instance, Quality of Service management [97], resource scheduling, and location tracking. Ontologies for pervasive computing environments will require logical models that include spatial and temporal logic, geometry, and other quantitative reasoning.

More fundamentally, Description Logics are not suited for some critical aspects of ubiquitous computing. Description Logic (DL) (also know as Terminological Logic) can reason about names, which can include objects and relations. DL does not deal with quantitative concepts; including order, quantity, time, or rates. Unfortunately, this kind of reasoning is essential to certain aspects of ubiquitous computing, including, for instance, Quality of Service management, resource scheduling, and location tracking. Future research should seek to extend DAML+OIL with additional logical models from spatial and temporal logic, geometry, and so on.

The DAML+OIL language is inadequate in describing concepts that deal with time, space, quantities, probabilities and certain other concepts. It might be useful to extend DAML+OIL so that such concepts can also be described within the same umbrella as terminological hierarchies. At the same time, issues of performance and decidability come into play while developing extensions. One of the powerful points in favor of description logics is that it is completely decidable, even though it may be too simple and limited for some purposes. So, there is a case in favor of not extending DAML+OIL to help it keep these properties. Other languages and logics would then have to be used to describe concepts involving time, quantities or probabilities. These issues will require further research in the future.

6.1.2. Development of Ontologies

Ontology development is not yet integrated with software development. The OILed tool and other similar tools (such as Protégé [50]) simplify the creation of ontologies for the pervasive computing environment. However, the deciding the contents of an ontology is still "Knowledge Engineering", and creating even a simple ontology is a challenging intellectual effort.

The OILed tool and other similar tools (such as Protégé [50]) simplify the creation of an ontology. However, the deciding the contents of an ontology is still "Knowledge Engineering", and even simple concepts can be represented more than one way. While this may not matter for a self-contained system, relatively minor differences in expression of the same concept can make two ontologies difficult to use together.

For example, consider the concept of a Web page which is identified by a URL. This can be modeled several ways, such as:

```
Class URL
   type:string
Class WebPage
    type: text
    url_of: URL
```

or, alternatively

```
Class
WebPage
    type: text
    URL:string
```

These two definitions are essentially the same, but are very difficult to automatically map to each other. It would be very useful to define standards, patterns, and tools for creating "standard interoperable" ontologies.

6.1.3. Standard Interfaces and APIs for Semantic Services

In our study, we created a CORBA service that imports and exports DAML+OIL XML, and defines a generic interface for a Knowledge Base. The Ontology Server is available to any service or application in the GAIA environment, which has enabled us to experiment with different uses of ontologies within the system. This experience shows the strong advantage that would emerge from the availability of a standard API for DAML+OIL (or preferably, OWL), and a standard CORBA interface for Knowledge Base services. The latter should use a standard XML-language, but be implemented by alternative logic engines and Knowledge Bases, such as Protégé [49], CLASSIC [45], OntoMerge [55] or a new version of FaCT [33, 57].

6.1.4. Scalability and Reliability of Semantic Services

The Pervasive Computing Environment is a long-running, open, real-time system. Maintaining an ontology in real-time as the system evolves presents important challenges for the design and implementation of ontologies and Knowledge Bases. In particular, the system needs to address the issues of:

- Large scale (many thousands of concepts and relations), many hundreds of services using the ontology and KB).
- Federation across autonomous locales
- Incremental updates (add, delete, or modify a few concepts in a large, active KB).
- Persistence and fault-tolerance

As discussed above, in this prototype implementation, the KB managed by the Ontology Server only has class information: the types or classes of different entities or terms, not descriptions of actual instances of entities (i.e., the current state of the system). This class information is sufficient for carrying out most of the tasks we are interested in (which will be described in the following sections).

A KB of description of instances would be far more dynamic than description of classes. Since instances can enter and leave the environment at any time, the Knowledge Base may have to be continuously updated to keep it synchronized with the current state of the space. Also, there potentially may be a very large number of instances of entities (compared to the number of classes). It would be very challenging to implement a KB that could efficiently handle large numbers of updates in real time. A KB of instances also requires a naming scheme so instances can be reliably recognized and distinguished, and would need robust error handling and recovery. The KB of class information is smaller and less volatile, so it could be implemented.

The information about existing entities is managed by other components of GAIA. GAIA has a service called the Space Repository which maintains information about the entities in the space at any time. Each entity has an XML description which is written in accordance to the meta-information about the entity as described in the ontology. The Space Repository maintains the descriptions of all entities that are currently in the space. More details about the Space Repository can be found in [63]. Instances of context information are distributed among different sensors and other entities that use context. The Ontology Server manages a unified ontology so that these distributed services can interoperate.

6.2. Future Research

6.2.1. Semantic interoperability between different environments

Different pervasive environments use their own set of ontologies. So, to enable entities in two different environments need to interact with each other, we need to establish some common semantic ground to enable correct interaction. This common semantic ground takes the form of a shared ontology that includes concepts in the ontologies of both the environments along with bridge concepts that relate concepts in the two sets of ontologies together.

Pervasive environments are inherently very dynamic and need to support mobility of entities. Thus, new entities can enter or leave these environments at any time. If the entities use different ontologies to describe their concepts, they make use of axioms which describe how concepts in one ontology are related to concepts in the other ontology. This allows new entities to enter the environment and take part in it seamlessly.

One way of tackling the problem is by using a shared upper ontology under which other ontologies can be attached. This will require improved "Knowledge Engineering" environments, which is an area of active research [14, 15, 43, 45, 67].

Merging ontologies from multiple autonomous sources is critical for the Pervasive Computing Environment. In particular, it is necessary to merge in descriptions of new classes of entities (devices, services, components, and so on) and context information (e.g., new sensors) as they are introduced. It should be possible to develop frameworks and editors to assure that the creators of new entities can create descriptions that can be easily merged into system ontologies.

In recent work, the DAML-S profile has been proposed [1, 60]. The profiles seeks to define the "Upper Ontology for Services," which will be "grounded" in specific Web Service description, e.g., using WSDL. In future work, we will extend this concept to explore ontological descriptions of different tasks in the Ubiquitous Computing Environment. This would allow us to map abstract tasks (like playing music) into appropriate concrete implementations of these tasks (like using a specific application to play the music and headphones to listen to the music). The mapping function would make use of ontological descriptions of tasks and entities, as well as the current context.

6.2.3. Vocabulary Translation

Description Logics can be useful for vocabulary mapping—translating similar concepts with different names (e.g., [38, 65, 67]). For example, consider the ontology for MP3 files, which might be defined to have properties "artist", "label", and so on. In a library, the MP3 file would be a sub-class of "library resource" (e.g., the Dublin Core standard (*dces*) [72]), with properties "creator", "publisher", and so on. It is likely that we would like to declare that our MP3 class is equivalent to the appropriate library resource, and that the property *MP3.artist* is equivalent to *dces.creator*, *MP3.label* is equivalent to *dces.publisher*, and so on.

These relations can be asserted as DAML *axioms*. For example, the MP3 class from the GAIA ontology can be declared to be the same class as Recording from the library ontology:

```
<daml:Class rdf:about="http://somewhere.net/GAIA/mp3.daml#MP3">
  <daml:sameClassAs>
    <daml:Class rdf:about="http://library.net/resources.daml#Recording"/>
  </daml:sameClassAs>
</daml:Class>
```

A more complex declaration could declare the logical equivalence of the properties.

This study did not consider how to discover or develop mappings between vocabularies, which is known to be extremely difficult (see, perhaps [21, 65, 67]). But the DAML+OIL XML language can be used to implement mappings when they are available.

6.2.3. Security and Access Control

Another area that requires investigation is security, privacy, and access control. The Semantic Web as a whole is largely conceived as a completely open system, in which everything is published for everyone to see. It is far from clear how access control could or should be applied, e.g., to the information in an ontology or a KB. Reasoning engines typically can't enforce security policies, and the DAML language, for instance, has no facility to limit visibility of concepts or attributes. This topic must be addressed in future research.

Acknowledgements

This research is supported in part by the National Science Foundation grant NSF 98-70736, NSF 9970139, and NSF infrastructure grant NSF EIA 99-72884.

Important aspects of this study used software from Iona [37] and University of Manchester [3, 54].

Listing 1

```
<daml:Class rdf:about="file:C:/ActiveSpaces/Semantics/MyOntology/ActiveSpace.daml#MP3Server">
 <rdfs:label>MP3Server</rdfs:label>
 <rdfs:comment><![CDATA[An MP3Server maintains a list of songs - this list can be searched by certain attributes and it can also be sent commands to play songs]]></rdfs:comment>
 <oiled:creationDate><![CDATA[2002-11-09T17:10:52Z]]></oiled:creationDate>
 <oiled:creator><![CDATA[ranganat]]></oiled:creator>
 <rdfs:subClassOf>
  <daml:Class rdf:about="file:C:/ActiveSpaces/Semantics/MyOntology/ActiveSpace.daml#SearchableService"/>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <daml:Class rdf:about="file:C:/ActiveSpaces/Semantics/MyOntology/ActiveSpace.daml#CommandableService"/>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <daml:Restriction>
   <daml:onProperty rdf:resource="file:C:/ActiveSpaces/Semantics/MyOntology/ActiveSpace.daml#executesDataType"/>
   <daml:hasClass>
    <daml:Class rdf:about="file:C:/ActiveSpaces/Semantics/MyOntology/ActiveSpace.daml#MP3File"/>
   </daml:hasClass>
  </daml:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <daml:Restriction>
   <daml:onProperty rdf:resource="file:C:/ActiveSpaces/Semantics/MyOntology/ActiveSpace.daml#searchableBy"/>
   <daml:hasClass>
    <daml:Class rdf:about="file:C:/ActiveSpaces/Semantics/MyOntology/ActiveSpace.daml#MP3Attributes"/>
   </daml:hasClass>
  </daml:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <daml:Restriction>
   <daml:onProperty rdf:resource="file:C:/ActiveSpaces/Semantics/MyOntology/ActiveSpace.daml#commandableBy"/>
   <daml:hasClass>
    <daml:Class rdf:about="file:C:/ActiveSpaces/Semantics/MyOntology/ActiveSpace.daml#MP3ServerPlay"/>
   </daml:hasClass>
  </daml:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <daml:Restriction>
   <daml:onProperty rdf:resource="file:C:/ActiveSpaces/Semantics/MyOntology/ActiveSpace.daml#commandableBy"/>
   <daml:hasClass>
    <daml:Class rdf:about="file:C:/ActiveSpaces/Semantics/MyOntology/ActiveSpace.daml#MP3ServerRandomPlay"/>
   </daml:hasClass>
  </daml:Restriction>
 </rdfs:subClassOf>
</daml:Class>
```

Listing 2

```
<daml:Class
        rdf:about="file:C:/ActiveSpaces/Semantics/MyOntology/file:C:\ActiveSpaces\Semantics\MyOntology\ActiveSpace.daml#
        Temperature">
<rdfs:label>TemperatureInformation</rdfs:label>
<rdfs:comment><![CDATA[]]></rdfs:comment>
<oiled:creationDate><![CDATA[2002-10-06T19:18:06Z]]></oiled:creationDate>
<oiled:creator><![CDATA[ranganat]]></oiled:creator>
<rdfs:subClassOf>
  <daml:Class rdf:about="file:C:/ActiveSpaces/Semantics/MyOntology/ActiveSpace.daml#WeatherInformation"/>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <daml:Restriction daml:cardinalityQ="1">
    <daml:onProperty rdf:resource="file:C:/ActiveSpaces/Semantics/MyOntology/ActiveSpace.daml#subject"/>
    <daml:hasClassQ>
     <daml:Class>
       <daml:unionOf>
         <daml:List>
          <daml:first>
            <daml:Class rdf:about="file:C:/ActiveSpaces/Semantics/MyOntology/ActiveSpace.daml#PhysicalPlace"/>
          </daml:first>
          <daml:rest>
           <daml:List>
           <daml:first>
            <daml:Class rdf:about="file:C:/ActiveSpaces/Semantics/MyOntology/ActiveSpace.daml#Person"/>
           </daml:first>
           <daml:rest>
             <daml:nil/>
           </daml:rest>
           </daml:List>
           </daml:rest>
         </daml:List>
       </daml:unionOf>
      </daml:Class>
    </daml:hasClassQ>
  </daml:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <daml:Restriction daml:cardinalityQ="1">
    <daml:onProperty rdf:resource="file:C:/ActiveSpaces/Semantics/MyOntology/ActiveSpace.daml#relator"/>
    <daml:hasClassQ>
        <daml:Class rdf:about="file:C:/ActiveSpaces/Semantics/MyOntology/ActiveSpace.daml#ComparisonOperator"/>
    </daml:hasClassQ>
  </daml:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <daml:Restriction daml:cardinalityQ="1">
    <daml:onProperty rdf:resource="file:C:/ActiveSpaces/Semantics/MyOntology/ActiveSpace.daml#object"/>
    <daml:hasClassQ>
       <daml:Class rdf:about="file:C:/ActiveSpaces/Semantics/MyOntology/ActiveSpace.daml#TemperatureValue"/>
    </daml:hasClassQ>
  </daml:Restriction>
</rdfs:subClassOf>
  </daml:Class>
```

References

1. Ankolekar, Anupriya, Burnstein, Mark, Hobbs, Jerry R., Lassila, Ora, Martin, David, McDermott, Drew, McIlraith, Sheila A., Narayanan, Srini, Paolucci, Massimo, Payne, Terry, and Sycara, Katia, "DAML-S: Web Service Description for the Semantic Web," *First International Semantic Web Conference (ISWC)*, Sardinia, 2002.
2. Bechhofer, Sean, Goble, Carole, and Horrocks, Ian, "DAML+OIL is not enough," *Second International Workshop on the Semantic Web*, Stanford, 2001.
3. Bechhofer, Sean, Horrocks, Ian, Patel-Schneider, Peter F., and Tessaris, Sergio, "A proposal for a description logic interface," *International Workshop on Description Logics (DL'99)*, Las Vegas, 1999.
4. Bechhofer, Sean, Horrocks, Ian, and Tessaris, Sergio, "CORBA interface for a DL Classifier," 1999.
5. Berners-Lee, T., Fielding, R., and Masinter, L., "Uniform Resource Identifiers (URI): Generic Syntax," IETF , RFC 2396 , 1998. http://www.ietf.org/rfc2398.txt
6. Berners-Lee, Tim, Hendler, James, and Lassila, Ora, "The Semantic Web," *Scientific American*, vol. 284, no. 5, pp. 35-43, 2001. http://www.sciam.com/2001/0501issue/0501berners-lee.html
7. Bryson, Joanna J., Martin, David L., McIlraith, Sheila A., and Stein, Lynn Andreas, "Toward Behavioral Intelligence in the Semantic Web," *IEEE Computer*, vol. 35, no. 11, pp. 48-54, 2002.
8. Chakraborty, Dipanjan, Perich, Filip, Avancha, Sasikanth, and Joshi, Anupam, "DReggie: Semantic Service Discovery for M-Commerce Applications," *Symposium on Reliable Distributed Systems*, 2001. http://daml.umbc.edu/papers/dreggie.pdf
9. daml.org, "The DARPA Agent Markup Language Homepage," http://www.daml.org.
10. daml.org, "Ontologies," http://www.daml.org/ontologies/.
11. Decker, Stefan, Fensel, Dieter, Harmelen, Frank van, Horrocks, Ian, Melnik, Sergey, Klein, Michel, and Broekstra, Jeen, "Knowledge Representation on the Web," *International Workshop on Description Logics*, 2000. http://www.ontoknowledge.org/iol/downl/DL00-oil.pdf
12. Dey, Anind K., "A Conceptual Framework and a Toolkit for Supporting Rapid Prototyping of Context-Aware Applications," *Human-Computer Interaction*, vol. 16, , 2001.
13. Edwards, W. Keith, *Core JINI*. Upper Saddle River, NJ: Prentice Hall, 1999.
14. Fensel, Dieter, *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Berlin: Springer, 2001.
15. Fensel, Dieter, "Ontology-Based Knowledge Management," *IEEE Computer*, vol. 35, no. 11, pp. 56-59, 2002.
16. Fensel, Dieter, Horrocks, Ian, Harmelen, Frank Van, Decker, Stefan, Erdmann, M., and Klein, Michel, "OIL in a Nutshell," *European Knowledge Acquisition Conference*, 2000. http://www.cs.vu.nl/~ontoknow/oil/downl/oilnutshell.pdf
17. Fensel, Dieter, Horrocks, Ian, Harmelen, Frank van, McGuinness, Deborah L., and Patel-Schneider, Peter F., "OIL: An Ontology Infrastructure for the Semantic Web," *IEEE Intelligent Systems*, vol. 16, no. 2, pp. 38-45, 2001.

18. Fikes, Richard and McGuinness, Deborah I., "An Axiomatic Semantics for RDF, RDF-S, and DAML+OIL," http://www.daml.org/2001/03/axiomatic-semantics-071601.html.
19. Franconi, Enrico, "Description Logics and Logics," http://www.cs.man.ac.uk/~franconi/dl/course.
20. Franconi, Enrico, "Propositional Description Logics," http://www.cs.man.ac.uk/~franconi/dl/course/propositional-dl.ps.gz.
21. Furnas, George W., Deerwester, Scott C., Dumais, Susan T., Landauer, Thomas K., Harshman, Richard A., Streeter, Lynn A., and Lochbaum, Karen E., "Information Retrieval Using A Singular Value Decomposition Model of Latent Semantic Structure," *Eleventh International ACM/SIGIR Conference on Research and Development in Information Retrieval*, Grenoble, 1988.
22. Gonzalez-Castillo, Javier, Trastour, David, and Bartolini, Claudio, "Description Logics for Matchmaking Services," HP Laboratories Bristol , Bristol , HPL-2001-265 , 2002. http://www.hpl.hp.com/techreports/2001/HPL-2001-265.html
23. Guarino, Nicola, "Formal Ontology and Information Systems," *Formal Ontology and Information Systems*, Trento, IT, 1998. http://www.ladseb.pd.cnr.it/infor/Ontology/Papers/FOIS98.pdf
24. Harmelen, Frank van, Patel-Schneider, Peter F., and Horrocks, Ian, "A Model-Theoretic Semantics for DAML+OIL," http://www.daml.org/2001/03/model-theoretic-semantics.html.
25. Harmelon, Frank, Pael-Schneider, Peter F., and Horrocks, Ian, "Annotated DAML+OIL (March 2001) Ontology Markup," http://www.daml.org/2001/03/daml+oil-walkthru.html.
26. Harmelon, Frank van, Patel-Schneider, Peter F., and Horrocks, Ian, "Reference description of the DAML+OIL (March 2001) ontology markup language," http://www.daml.org/2001/03/reference.html.
27. Hendler, James, "Agents and the Semantic Web," *IEEE Intelligent Systems*, vol. 16, no. 2, pp. 30-37, 2001.
28. Hoffman, Kevin, Gabriel, Jeff, Gosnell, Denise, Hasan, Jeff, Holm, Cristian, Musters, Ed, Narkiewickz, Jan, Schenken, John, Thangarathinam, Thiru, Wylie, Scott, and Ortiz, Jonothan, *Professional .NET Framework*. Birmingham: WROX Press Ltd., 2001.
29. Horrocks, Ian, "CORBA-FaCT," http://www.cs.man.ac.uk/~horrocks/FaCT/CORBA-FaCT.html.
30. Horrocks, Ian, "A Denotational Semantics for Standard OIL and Instance OIL," http://www.ontoknowledge.org/oil/downl/semantics.pdf.
31. Horrocks, Ian, "The FaCT system," *Automated Reasoning with Analytic Tableaux and Related Methods*, 1998.
32. Horrocks, Ian, "Reasoning with Expressive Description Logics: Theory and Practice," : University of Leipzig, 2001. http://www.cs.man.ac.uk/~horrocks/Slides/leipzig-jun-01.pdf
33. Horrocks, Ian and Sattler, Ulrike, "Ontology Reasoning with SHOQ(D) Description Logic," *International Joint Conference on Artificial Intelligence*, Seattle, 2001.

34. Horrocks, Ian, Sattler, Ulrike, and Tobias, Stephan, “Practical Reasoning for Expressive Description Logics,” *International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, Tbilisi, 1999.
35. Horrocks, Ian and Tessaris, Sergio, “Querying the Semantic Web: A Formal Approach,” *International Semantic Web Conference*, Sardinia, 2002.
36. IBM, "IBM WebSphere SDK for Web Services (WSDK) Version 5.0," http://www-106.ibm.com/developerworks/webservices/wsdk/.
37. IONA Technologies Inc., "ORBacus Trader, Version 2.0.0," http://www.iona.com/products/orbacus_trader.html.
38. Lancaster, F. W., *Vocabulary Control for Information Retrieval*. Arlington, VA: Information Retrieval Press, 1986.
39. Lyytinen, Kalle and Yoo, Youngjin, “Issues and Challenges in Ubiquitous Computing,” *Communications of the ACM*, vol. 45, no. 12, pp. 62-65, 2002.
40. Maedche, Alexander and Staab, Stephen, “Ontology Learning for the Semantic Web,” *IEEE Intelligent Systems*, vol. 12, no. 2, pp. 72-79, 2001.
41. McGrath, Robert E., “Discovery and Its Discontents: Discovery Protocols for Ubiquitous Computing,” Department of Computer Science University of Illinois Urbana-Champaign , Urbana , UIUCDCS-R-99-2132 , March 25 2000.
42. McGrath, Robert E., *A Model for Physical Objects in a Ubiquitous Computing Environment (Ph. D thesis, to appear)*, Ph. D. Thesis in Computer Science, University of Illinois, Urbana-Champaign, Urbana, 2003.
43. McGuinness, Deborah L., “Conceptual Modeling for Distributed Ontology Environments,” *International Conference on Conceptual Structures, Logical, Linguistic, and Computational Issues*, Darmstadt, 2000.
44. McGuinness, Deborah L., “Ontologies and Online Commerce,” *IEEE Intelligent Systems*, vol. 16, no. 1, pp. 8-14, 2001.
45. McGuinness, Deborah L., Fikes, Richard, Rice, James, and Wilder, Steve, “An Environment for Merging and Testing Large Ontologies,” *International Conference on Principles of Knowledge Representation and Reasoning*, Breckenridge, CO, 2000.
46. McIlraith, Sheila A., Son, Tran Cao, and Zeng, Honlei, “Semantic Web Services,” *IEEE Intelligent Systems*, vol. 16, no. 2, pp. 46-53, 2001.
47. Microsoft, "XML Web Services Developer Center Home," http://msdn.microsoft.com/wevservices/index.aspx.
48. Minsky, Marvin, “A Framework for Representing Knowledge,” in *The Psychology of Computer Vision*, Winston, P., Ed. New York: McGraw Hill, 1975.
49. Noy, Natalya Fridman, Fergerson, Ray W., and Musen, Mark A., “The Knowledge Model of Protege-2000: Combining Interoperability and Flexibility,” *Twelfth International Conference on Knowledge Engineering and Knowledge Management*, 2000.
50. Noy, Natalya F., Sintek, Michael, Decker, Stefan, Crubezy, Monica, Fergerson, Ray W., and Musen, Mark A., “Creating Semantic Web Contents with Protege-2000,” *IEEE Intelligent Systems*, vol. 16, no. 2, pp. 60-71, 2001.
51. Object Management Group, “CORBAservices: Common Object Services Specification,” Object Management Group , 1999. ftp://ftp.omg.org/pub/.docs/formal/98-07-05.pdf

52. Object Management Group, "TC Plenaries and Subgroup Directory," http://www.omg.org/technology/documents/domain_spec_catalog.htm.
53. Object Management Group, "Trading Service Specification," 2000.
54. OilEd, "OilEd," http://oiled.man.ac.uk/.
55. OntoMerge, "OntoMerge: Ontology Translation by Merging Ontologies," http://cs-www.cs.yale.edu/homes/dvm/daml/ontology-translation.html.
56. Orfali, Robert and Harkey, Dan, *The Essential Distributed Objects Survival Guide*. New York: John Wiley and Sons, Inc., 1996.
57. Pan, Jeff Z. and Horrocks, Ian, "Reasoning in the SHOQ(D) Description Logic," *Workshop on Description Logics (DL-2002)*, 2002. http://dl-web.man.ac.uk.~panz/Zhilin/download/Papers/Pan-Horrocks-shoqdn-2002.pdf
58. Pascoe, Bob, "Salutation Architectures and the newly defined service discovery protocols from Microsoft and Sun," Salutation Consortium , White Paper June 6 1999. http://www.salutation.org/whitepaper/JINI-UPnP
59. Payne, Terry R., Singh, Rahul, and Sycara, Katia, "RCal: A Case Study on Semantic Web Agents," *First International Conference on Autonomous Agents and Multi-Agent Systems*, 2002.
60. Peer, Joachim, "Bringing Together Semantic Web and Web Services," *First International Semantic Web Conference*, Sardinia, 2002.
61. Quillian, M. Ross, "Semantic Networks," in *Semantic Information Processing*, Minsky, Marvin, Ed. Cambridge: MIT Press, 1968.
62. Reynolds, Dave, "Semantic Web Chalk Talk: Amateur Intro to Decription Logics," . Bristol: HP Laboratories, 2001. http://www.hpl.hp.com/semweb/dwonload/DescriptionLogicsIntro.pdf
63. Roman, Manuel, Hess, Christopher K., Cerqueira, Renato, Ranganathan, Anand, Campbell, Roy H., and Nahrstedt, Klara, "GAIA: A Middleware Infrastructure to Enable Active Spaces," *IEEE Pervasive Computing*, vol. 1, no. 4, pp. pp. 74-83, 2002.
64. Sadeh, Norman, Chan, Enoch, Shmazaki, Yoshinori, and Van, Linh, "MyCampus: An Agent-Based Environment for Context-Aware Mobile Services," *Workshop on Ubiquitous Agents on Embedded, Wearable, and Mobile Devices*, Bologna, 2002.
65. Schatz, Bruce, "Information Retrieval in Digital Libraries: Bringing Search to the Net," *Science*, vol. 275, pp. 327-334, 1997.
66. semanticweb.org, "Markup Languages and Ontologies," http://www.semanticweb.org/knowmarkup.html.
67. Stuckenschmidt, Heiner, Harmelen, Frank van, Fensel, Dieter, Klein, Michel, and Horrocks, Ian, "Catalogue Integration: A Case Study in Ontology-Based Semantic Translation," 2000. http://www.ontoknowledge.org/oil/downl/CatIntegr.pdf
68. Sun Microsystems, "Java Remote Method Invocation (RMI)," http://java.sun.com/products/jdk/1.2/guide/rmi/index.html.
69. Sun Microsystems Inc., "A Collection of Jini (TM) Technology Helper Utilities and Services Specifications: Version 1.2," December 2001.
70. Sycara, K., Paolucci, M., Velsen, M. van, and Giampapa, J., "The RETSINA MAS Infrastructure," Carnegie Mellon University , Robotics Institute Technical Report CMU-RI-TR-01-05 , 2001.

71. Tamma, Valentina, Wooldridge, Michael, and Dickinson, Ian, “An ontology based approach to automated negotiation,” *Proceedings of the IV workshop on agent mediated electronic commerce (AMEC IV)*, Bologna, 2002.
72. The Dublin Core Metadata Initiative, "Dublin Core Metadata Initiative - Home Page," http://purl.org/dc/index.htm.
73. Trastour, David, Bartolini, Claudio, and Gonzalez-Castillo, Javier, “A Semantic Web Approach to Service Description for Matchmaking of Services,” HP Laboratories Bristol , Bristol , HPL-2001-183 , July 30 2001. http://www.hpl.hp.com/techreports
74. Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S., Kesselman, C., and Vanderbilt, P., “Grid Service Specification,” BWD-R October 4 2002. http://www.ggf.org/meetings/ggf6_wg_papers/draft-ggf-gridservice-04_2002-10-04.pdf
75. uddi.org, “UDDI Technical White Paper,” September 6 2002. http://www.uddi.org/pubs/UDDI_Executive_White_Paper.pdf
76. uddi.org, “UDDI Version 3.0,” 19 July 2002. http://uddi.org/pugs/uddi-v300-published-20020719.pdf
77. United Nations, "United Nations Standard Produce and Services Code," hhtp://www.un-spsc.net/.
78. W3C, "Extensible Markup Language (XML)," http://www.w3.org/XML.
79. W3C, “Feature Synopsis for OWL Lite and OWL,” W3C Working Draft 29 July 2002. http://www.w3.org/TR/2002/WD-owl-features-20020729/
80. W3C, "Namespaces in XML," http://www.w3.org/TR/REC-xml-names/.
81. W3C, "Requirements for a Web Ontology Language," http://www.w3c.org/TR/webont-req/.
82. W3C, "Resource Description Framework (RDF)," http://www.w3c.org/RDF.
83. W3C, "The Semantic Web," http://www.w3.org/2001/sw.
84. W3C, “SOAP Version 1.2 Part 1: Messaging Framework,” W3C Candidate Recommendation 19 December 2002. http://www.w3.org/TR/soap12-part1/
85. W3C, “Web Ontology Language (OWL) Guide Version 1.0,” W3C Working Draft 4 November 2002. http://www.w3.org/TR/2002/WD-owl-guide-20021104/
86. W3C, “Web Ontology Language (OWL) Reference Version 1.0,” 12 November 2002. http://www.w3.org/TR/2002/WD-owl-ref-20021112/
87. W3C, “Web Services Architecture,” W3C Working Draft 14 November 2002. http://www.w3.org/TR/2002/WD-ws-arch-20021114/
88. W3C, “Web Services Architecture Usage Scenarios,” W3C Working Draft 30 July 2002. http://www.w3.org/TR/2002/WD-ws-arch-scenarios-20020730/
89. W3C, “Web Services Description Language (WSDL) Version 1.2,” W3C Working Draft 9 July 2003. http://www.w3.org/TR/wsdl12/
90. W3C, “Web Services Description Language (WSDL) Version 1.2: Bindings,” W3C Working Draft 9 July 2002. http://www.w3.org/TR/wsdl1.2-bindings/
91. W3C, “Web Services Description Requirements,” W3C Working Draft 28 October 2002. http://www.w3.org/TR/ws-desc-reqs/
92. W3C, “Web Services Description Usage Scenarios,” W3C Working Draft 4 June 2002. http://www.w3.org/TR/ws-desc-usecases/
93. W3C, "Web-Ontology (WebOnt) Working Group," http://www.w3.org/2001/sw/WebOnt.

94. W3C, "XML Schema," http://www.w3.org/XML/Schema.html.
95. W3C, "XML Schema Part 2: Datatypes," http://www.w3c.org/TR/xml-schema-2.
96. Wahl, M., Howes, T., and Kille, S., “Lightweight Directory Access Protocol (v3),” IETF , RFC 2251 , December 1997. http://www.rfc-editor.org/rfc/rfc2251.txt
97. Wichadakul, Duangdao, Gu, Xiaohui, and Nahrstedt, Klara, “A Programming Framework for Quality-Aware Ubiquitous Multimedia Applications,” *ACM Multimedia*, Juan Les Pins, 2002.