Chapter 20
An Ontology-Driven Approach and a Context Management Framework for Ubiquitous Computing Applications

Christos Goumopoulos and Achilles Kameas

20.1 Introduction

Pervasive or Ubiquitous computing is a new technological paradigm in which every thing around us has built-in and interconnected computers (Weiser, 1991; Disappearing Computer, 2007). Embedded in everyday objects these interconnected devices (also called artifacts) open up an unlimited number of possibilities for many new applications and services (Norman, 1999; Bergman, 2000). Applications result from the dynamic and adaptive composition of such artifacts, triggered via explicit user/application requests, application/task templates, or even more autonomic interaction schemes. Then, in this context, a “system” is defined to be the collective, complex service that emerges as an aggregation of simpler services offered by independent artifacts.

Ontologies can help address some key issues of Ubiquitous computing environments such as knowledge representation, semantic interoperability and service discovery. One important issue, for example, to be resolved in building a Ubiquitous computing system, is the development of interfaces between heterogeneous and incompatible components, objects or artifacts. This can be dealt with by developing an ontology of concepts so that different types of functionality and interactions or artifact bindings can be described in a way that is natural and consistent across different systems. If the services provided by artifacts are to be properly exploited, then it must be ensured that they will be able to interpret the representations sent to them and to generate the representations expected from them. Following a service-oriented approach, applications state their requirements in terms of concepts that are part of the application’s ontology rather than specific resource instances.

It is also important to capture complex interactions between many different artifacts. Thus, apart from simple peer-to-peer interactions, appropriate descriptions are needed to support more complex interaction structures; both synchronous

C. Goumopoulos (✉)
Distributed Ambient Information Systems Group, Hellenic Open University & Computer Technology Institute, Patras, Hellas, Greece
e-mail: goumop@cti.gr

and asynchronous schemes are required to cater for the complexity of pervasive computing applications. To achieve synergy, the system has to apply global decision making procedures in order to cope with distributed resource management, service composition and performance optimization. At the same time, each artifact employs local decision making procedures in order to adjust its autonomous operation to changes of context, to learn and to maintain its stable operation.

The goal of this chapter is to present an ontology-driven approach and a context management framework for the composition of context-aware Ubiquitous computing applications. The next section outlines how context is modeled and used in various Ubiquitous computing systems emphasizing on ontology-oriented approaches. Then we describe the ontology that was developed in order to conceptually represent context-aware Ubiquitous computing systems. This ontology is designed taking into account both the autonomous nature of components, objects and artifacts and the necessity of their interoperability; so it is divided into two layers, a private (application-specific) and a common (core) one. The core ontology defines a meta-model of the Ubiquitous computing domain based on the Bunge-Wand-Weber (BWW) ontology. Then we present a hierarchical approach for engineering context-aware Ubiquitous computing systems including the context management and decision-making processes as well as the analysis of the mechanism that was developed based on that processes. Finally, we conclude by evaluating our ontology-driven approach and presenting the lessons learned. A prototype application is also outlined where an augmented plant is incorporated in a Ubiquitous computing environment in order to collaborate with other augmented objects, providing thus a communication channel between plants and people.

20.2 Ontology Based Modeling of Context Aware Ubiquitous Computing Systems

According to (Dey, 2001) context is: "*Any information that can be used to characterize the situation of entities (i.e. whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity and state of people, groups and computation and physical objects.*" In Ubiquitous computing applications different kinds of context can be used like physical (e.g. location and time), environmental (e.g. weather and light) and personal information (e.g. mood and activity). Nevertheless, the term context mostly refers to information relative to location, time, identity and spatial relationships.

A number of informal and formal context models have been proposed in various Ubiquitous computing systems; a survey of context models is presented in (Strang and Linnhoff-Popien, 2004). Among systems with informal context models, the Context Toolkit (Dey et al., 2001) represents context in form of attribute-value tuples. The Cooltown project (Kindberg et al., 2000) proposed a Web based model of context in which each object has a corresponding Web description. Both ER and UML models are used for the representation of formal context models by (Henricksen et al., 2002).

As ontologies are a promising instrument to specify concepts and their interrelations, they can provide a uniform way for specifying the core concepts of a context model, as well as an arbitrary amount of subconcepts and facts, altogether enabling contextual knowledge sharing and reuse in a Ubiquitous computing system (De Bruijn, 2003). Thus several research groups have presented ontology-based models of context and used them in Ubiquitous computing applications. In the following, we briefly describe the most representative ones.

In the GAIA framework (Ranganathan and Campbell, 2003), an infrastructure is presented that supports the gathering of context information from different sensors and the delivery of appropriate context information to Ubiquitous computing applications. The project aim was to develop a flexible and expressive model for context able to represent the wide variety of possible contexts and to support complex reasoning on contexts. Context is represented with first-order predicates written in DAML+OIL. This context model allows deriving new context descriptions from other sensed context.

GLOSS (GLObal Smart Space) is a software infrastructure that enables the interactions between people, artifacts, and places, while taking account of both context and movement on a global scale (Dearle et al., 2003). By exploiting the features of physical spaces, it uses people's location and movement as a source of task-level context and as a guide to provide appropriate information, or services. Therefore, GLOSS facilitates the low-level interactions (such as tracking a user's location) that are driven by high-level contexts (such as a user's task). This system accommodates both service heterogeneity and evolution, using ontologies. The GLOSS ontologies describe concepts, which provide the precise understanding of how services (physical and informational) are used and how users interleave various contexts at run time.

CoBrA (Context Broker Architecture) is a pervasive context-aware computing infrastructure that enables Ubiquitous agents, services and devices, to behave intelligently according to their situational contexts (Kagal et al., 2001). It is a broker-centric agent architecture that provides knowledge sharing, context reasoning, and privacy protection support for Ubiquitous context-aware systems, using a collection of ontologies, called COBRA-ONT, for modeling the context in an intelligent meeting room environment (Chen et al., 2003). These ontologies are expressed in the Web Ontology Language (OWL), define typical concepts associated with places, agents, and events and are mapped to the foundational ontologies that are relevant to smart spaces.

Wang et al. created an upper ontology, the CONON (Wang et al., 2004) context ontology, which captures general features of basic contextual entities, a collection of domain specific ontologies and their features in each subdomain. The upper ontology is a high-level ontology which defines basic concepts about the physical world such as "person", "location", "computing entity", and "activity". The domain-specific ontologies, are a collection of low-level ontologies, which define the details of general concepts and their properties in each sub-domain where they apply to (like home domain, office domain). All these context ontologies help in sharing a common understanding of the structure of contextual information coming from users, devices, and services, so as to support semantic interoperability and reuse of domain knowledge. The CONON ontologies are serialized in OWL-DL which has a semantic equivalence to the well researched description logic (DL). Thus CONON supports two types of reasoning: reasoning to detect and correct inconsistent context information and reasoning as a means to derive higher level context information. The latter type of reasoning is based on properties like symmetry and transitivity, as well as on user-defined rules. The CONON ontology is part of the SOCAM (Service-Oriented Context-Aware Middleware) architecture, which supports the building and rapid prototyping of context-aware services in pervasive computing environments (Gu et al., 2004).

The Context Ontology Language (CoOL) (Strang et al., 2003) is based on the Aspect-Scale-Context Information (ASC) model. Aspects represent classifications (e.g. Temperature), while scales are individual dimensions of aspects (e.g. Celsius). Context information is attached to a particular aspect and scale; quality metadata (e.g. meanError) is associated with information via quality properties. This contextual knowledge is evaluated using ontology reasoners, like F-Logic and OntoBroker. In addition to the determination of service interoperability in terms of contextual compatibility and substitutability, this language is used to support context-awareness in distributed service frameworks for various applications.

The CADO (Context-aware Applications with Distributed Ontologies) framework (De Paoli and Loregian, 2006) relies on distributed ontologies that are shared and managed in a peer-to-peer fashion. It is composed of three layers and designed to support mobility of workers in complex work settings. The three layers ensure semantic interoperability via the process of ontology merging, while context and application interoperability are ensured using Context and Interaction Managers respectively.

The CoCA (Collaborative Context-Aware) system is a collaborative, domain independent, context-aware middleware platform, which can be used for context-aware application development in Ubiquitous computing (Ejigu et al., 2007). This architecture for context-aware services focuses on context-based reasoning in Ubiquitous computing environments, using semantic-based collaborative approaches. The model uses an ontology for modeling and management of context semantics and a relational database schema for modeling and management of context data. These two elements are linked through the semantic relations built in the ontology.

The GAS Ontology (Christopoulou and Kameas, 2005) is based on a different approach for modeling Ubiquitous computing applications that are composed of artifacts. It adopts GAS, the Gadgetware Architectural Style (Kameas et al., 2003) according to which artifacts are called eGadgets, their services are called Plugs and the combination of two services is called a Synapse. GAS Ontology aims to conceptualize GAS by describing the semantics of these basic terms and by defining the relations among them. Thus the GAS Ontology provides a shared means for the communication and collaboration among artifacts, even though they may be produced by different manufacturers. This approach serves as the basis of our work that is presented later in this chapter.

Although each research group follows a different approach for using ontologies in modeling and managing context in Ubiquitous computing applications, it has been acknowledged by the majority of researchers (Dey, 2001; Henricksen et al., 2002; Ranganathan and Campbell, 2003; Christopoulou and Kameas, 2005) that it is a necessity to decouple the process of context acquisition and interpretation from its actual use. This can be achieved by introducing a consistent, reliable and efficient context framework which can facilitate the development of context-aware applications. In this respect, we propose an approach for a context-aware Ubiquitous computing system that eases the composition of such context-aware Ubiquitous computing applications and separates this process from the process of context acquisition.

The use of ontologies to model context-aware systems facilitates knowledge sharing across different systems and context reasoning based on semantic Web technologies. An important distinction between the approaches presented above and the one we adopted to develop the GAS ontology is that the former are based on understanding of ontology as a specification of some conceptualization (Guizzardi et al., 2002), whereas we approach ontology in philosophical terms, e.g. as in (Milton and Kazmierczak, 2004); this led to the development of an abstract meta-model for the Ubiquitous computing environment.

20.3 An Ontology-Driven Meta-Model for Ubiquitous Computing Systems

20.3.1 Underlying Concepts

From the system engineering perspective, conceptual modeling is at the core of systems analysis and design. Our approach for developing a conceptual model that represents the structural, relational and behavioral elements of the targeted Ubiquitous computing systems is based on the so-called Bunge-Wand-Weber (BWW) ontology. Ontology in this context represents a well-established theoretical domain within philosophy dealing with the models of reality. Wand and Weber have taken and extended an ontology presented by Mario Bunge (Bunge, 1977, 1979) and developed a formal foundation for modeling information systems (Wand and Weber, 1990). BWW Ontology has been widely applied in the information systems research field in contexts such as comparison of information systems analysis and design grammars, ontological evaluation of modeling grammars, information systems interoperability and for requirements engineering for commercial-off-the-shelf software and alignment in enterprise systems implementations (Rosemann et al., 2004).

Although the BWW ontology constructs have been originally defined using a rigorous set-theoretic language in many subsequent works the researchers attempted to simplify and clarify the explanation of the constructs by defining those using plain English (Weber, 1997). Following is the description of selected core ontological constructs of the BWW ontology:

- *Thing*: A thing is the basic construct in the BWW ontological model. The world is made of things that have properties. Two or more things (composite or primitive) can be associated into a *composite* thing.
- *Property*: We know about things in the world via their properties. A property is modeled via a function that maps the thing into some value. Properties are classified in a number of categories: *hereditary*, *emergent*, *intrinsic*, *binding/non-binding* and *mutual*.
- *Mutual Property*: A property that is meaningful only in the context of two or more things.
- *State*: The vector of values for all property functions of a thing is the state of the thing.
- *Conceivable State*: The set of all states that the thing may ever assume.
- *Stable state*: A stable state is a state in which a thing, subsystem, or system will remain unless forced to change by virtue of the action of a thing in the environment (an external event).
- *Transformation of a Thing*: A mapping from a domain comprising states to a co-domain comprising states.
- *System*: A set of things will be called a system, if, for any bi-partitioning of the set, interactions exist among things in any two subsets.
- *System Composition*: The things in the system are its composition.
- *System Environment*: Things that are not in the system, but which interact with things in the system are called the environment of the system.
- *System structure*: The set of couplings that exist among things within the system, and among things in the environment of the system and things in the system.
- *Subsystem*: A system whose composition and structure are subsets of the composition and structure of another system.

For developing a conceptual model for the Ubiquitous computing application domain, instead of using the entire BWW ontology, a more focused ontology is derived, by taking into consideration the requirements of the target application domain. Therefore, an appropriate subset of concepts is selected by applying elimination and specialization processes. In a similar perspective (Rosemann and Green, 2000), argue that taking into account the objectives of the modeling tasks in a specific problem domain as well as the type of users to be involved can assist in developing new ontologically based specific modeling grammars. In the next section we extend the concept of Thing to the concept of Entity and the concept of Composite Thing to the concept of Ambient Ecology. New concepts are introduced like the Plug and Synapse in order to provide detailed representation of the interaction among entities. The main advantage of this process is that the focused ontology is based on a well-established ontology with theoretical foundations. In addition, in order to communicate clearly and relatively easily the concepts of the derived conceptual model, we developed a description of the ontological constructs using a meta-model. Through this meta-model, the understanding of the ontological constructs and how they relate to each other can be explained clearly. We have used the UML meta-language for that purpose.

20.3.2 Focused Ontology

Our model defines the logical elements necessary to support a variety of applications in Ubiquitous computing environments. Its basic definitions are given below. A graphical representation of the concepts and the relations between them is given as a UML class diagram in Fig. 20.1.

eEntity: An eEntity is the programmatic bearer of an entity (i.e. a person, place, object, biological being or a composition of them). An eEntity constitutes the basic component of an Ambient Ecology. “e” stands here for extrovert. Extroversion is a central dimension of human personality, but in our case the term is borrowed to denote the acquired through technology competence of an entity to interact with other entities in an augmented way for the purpose of supporting the users’ everyday activities meaningfully. This interaction is mainly related with either the provision or consumption of context and services between the participating entities. A coffee maker, for instance, publishes its service to boil coffee, while context for a person may denote her activity and location. An augmented interaction between the coffee maker and the person is the activation of the coffee machine when the person awakes in the morning. For this to happen we need probably a bed instrumented with pressure sensors (an artifact) and a reasoning function for the persons’ awaking activity, which may not be trivial to describe. An eEntity in general possesses properties of three types: *structural* which belong to the entity itself; *relational* which relate the entity to other entities; and *behavioral* which determine possible changes to the values of structural and relational properties.

Artifacts: An artifact is a tangible object (biological elements like plants and animals are also possible) which bears digitally expressed properties; usually it is an object or device augmented with sensors, actuators, processing, networking unit etc. or a computational device that already has embedded some of the required

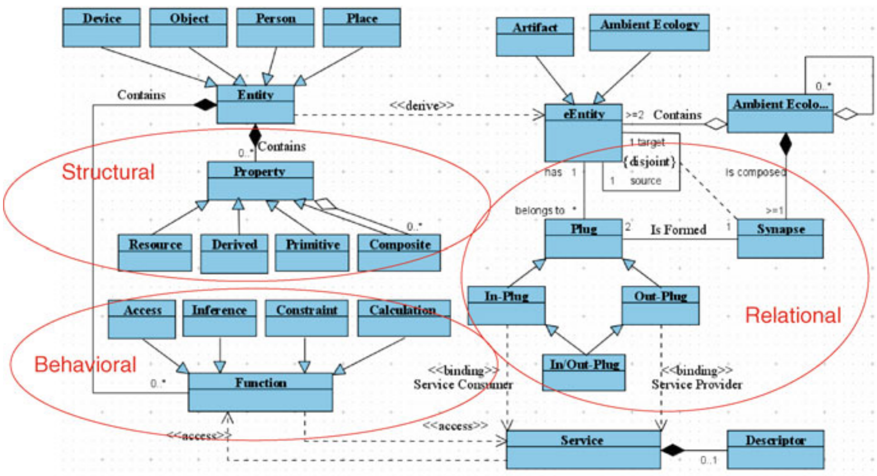

Fig. 20.1 A meta-model for the Ubiquitous computing Environment

hardware components. Software applications running on computational devices are also excessively considered to be artifacts. Examples of artifacts are furniture, clothes, air conditioners, coffee makers, a software digital clock, a software music player, a plant, etc.

Ambient Ecology: Two or more eEntities can be combined in an eEntity synthesis. Such syntheses are the programmatic bearers of Ambient Ecologies and can be regarded as service compositions; their realization can be assisted by end-user tools. Since the same eEntity may participate in many Ambient Ecologies the whole-part relationship is not exclusive. In the UML class diagram (see Fig. 20.1) this is implied by using the aggregation symbol (hollow diamond) instead of the composition symbol (filled diamond). Ambient Ecologies are synthesizable, because an Ambient Ecology is itself regarded as an eEntity and can participate in another Ambient Ecology.

Properties: eEntities have properties, which collectively represent their physical characteristics, capabilities and services. A property is modeled as a function that either evaluates an entity's state variable into a single value or triggers a reaction, typically involving an actuator. Some properties (i.e. physical characteristics, unique identifier) are entity-specific, while others (i.e. services) may be not. For example, attributes like color/shape/weight represent properties that all physical objects possess. The service light may be offered by different objects. A property of an entity composition is called an *emergent* property. All of the entity's properties are encapsulated in a *property schema* which can be send on request to other entities, or tools (e.g. during an entity discovery).

Functional Schemas: An entity is modeled in terms of a functional schema: $F = \{f_1, f_2, \ldots, f_n\}$, where each function f_i gives the value of an observed property i in time t. Functions in a functional schema can be as simple or complex is required to define the property. They may range from single sensor readings to rule-based formulas involving multiple properties, to first-order logic so that we can quantify over sets of artifacts and their properties.

State: The values for all property functions of an entity at a given time represent the state of the entity. For an entity E, the set $P(E) = \{(p_1, p_2 \ldots p_n) \,|\, p_i = f_i(t)\}$ represents the state space of the entity. Each member of the state vector represents a *state variable*. The concept of state is useful for reasoning about how things may change. Restrictions on the value domain of a state variable are then possible.

Services: Services are resources capable of performing tasks that form a coherent functionality from the point of view of provider entities and requester entities. Services are self-contained, can be discovered and are accessible through synapses. Any functionality expressed by a service descriptor (a signature and accessor interface that describes what the service offers, requires and how it can be accessed) is available within the service itself and is manifested by plugs.

Transformation: A transformation is a transition from one state to another. A transformation happens either as a result of an internal event (i.e. a change in the state of a sensor) or after a change in the entitys' functional context (as it is propagated through the synapses of the entity).

Plugs: Plugs represent the interface of an entity. An interface consists of a set of operations that an entity needs to access in its surrounding environment and a set of operations that the surrounding environment can access on the given entity. Thus, plugs are characterized by their direction and data type. Plugs may be output (O) in case they manifest their corresponding property (e.g. as a provided service), input (I) in case they associate their property with data from other artifacts (e.g. as service consumers), or I/O when both happens. Plugs also have a certain data type, which can be either a semantically primitive one (e.g. integer, boolean, etc.), or a semantically rich one (e.g. image, sound etc.). From the user's perspective, plugs make visible the entities' properties, capabilities and services to people and to other entities.

Synapses: Synapses are associations between two compatible plugs. In practice, synapses relate the functional schemas of two different entities. Whenever the value of a property of a source entity changes, the new value is propagated to the target entity, through the synapse. The initial change of value caused by a state transition of the source entity causes finally a state transition to the target entity. In that way, synapses are a realization of the functional context of the entity.

20.3.3 Core vs. Application Ontology

The ontology that supports the development of Ubiquitous computing applications is divided in two basic layers: the Core and the Application layer. The discussed approach is in line with the design criteria proposed in (Gruber, 1993) for efficient development of ontologies:

- *Maximum monotonic extensibility*: new general or specialized terms can be included in the ontology in such a way that it does not require the revision of existing definitions.
- *Clarity*: terms which are not similar (common-sense terms vs. specialized domain ontologies) are placed in different taxonomies.

Core ontology – represents core knowledge of the Ubiquitous computing environment. This layer is designed to ensure *syntactic* and *structural* interoperability between various artifacts. Since the core ontology describes the language that artifacts use to communicate and collaborate it must be the same for all artifacts. A key issue is that the core ontology cannot be changed and contains only the necessary information in order to be small. In that way even artifacts with limited memory capacity can store and have access to the basic definitions. A graphical representation of the core ontology is given in Fig. 20.1. The basic classes of the core ontology have been discussed in the previous section.

Application ontology – represents knowledge about the application environments such as specific type of users and artifacts and acquired knowledge through synapses. This layer is designed to ensure *semantic* interoperability. The knowledge represented by application ontology is described as instances of the classes defined in the core ontology. In that sense the application ontology is not a stand-alone ontology as it does not contain the definition of its concepts and their relations. The application ontology represents the description of each artifact that is related with an application containing information about physical properties, plugs and the services that are provided through these plugs. For example, the application ontology of the *eLamp* artifact contains knowledge about the physical characteristics of *eLamp*, such as luminosity, the description of a plug with an identifier "OnOff" based on the definition provided by core ontology as well as the declaration that this plug is associated with the service "light". As services are the primary constituents of Ubiquitous computing systems the application ontology must contain specific service descriptions in order to support the service discovery and invocation mechanism. The application ontology describes also the synapses, the plugs of the artifact that participate in synapses, and the information about the capabilities/services of other artifacts that has been acquired via the synapses. Contrary to core ontology, the size of which must be small, the size of the application ontology can be as large as required, bounded only by the artifacts' memory capacity. In addition, the application ontology is dynamic and can change over time without causing problems to artifact collaboration. The dynamic nature of the application ontology results from the knowledge that can be acquired through the various synapses that may be established between artifacts.

20.4 Context Management Framework

20.4.1 Context Management Process

A Ubiquitous computing application typically consists of an infrastructure used to capture context and a set of rules governing how the application should respond to changes in this context. In order to isolate the user from the process of context acquisition and management and on the other hand provide her with a Ubiquitous computing system that enables the composition of context-aware applications we propose that the system is organized in a hierarchy of levels.

The design approach for composing context-aware Ubiquitous computing applications needs to be backed by an engineering methodology that defines the correct formulation of the context and behavior. The proposed context management process is depicted in Fig. 20.2. The motivation for this process emerged from the fact that artifacts in Ubiquitous computing environment may be in different "states" that change according to the artifacts' use by users and their reaction is based both on users' desires and these states.

The first step in this context management process is the acquisition of the low level context, which is the raw data given by the sensors (Lexical Level). A set of sensors are attached to an artifact so that to measure various artifact parameters, e.g. the position and the weight of an object placed on an augmented table. As the output

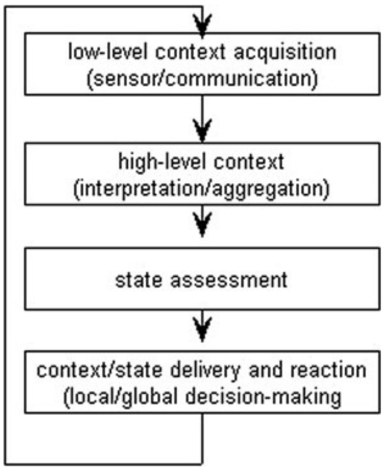
low-level context acquisition (sensor/communication)
high-level context (interpretation/aggregation)
state assessment
context/state delivery and reaction (local/global decision-making

Fig. 20.2 Context management process

of different sensors that measure the same artifact parameter may differ, e.g. sensors may use different metric system, it is necessary to interpret the sensors' output to higher level context information (Syntactical/Representation Level). Aggregation of context is also possible meaning that semantically richer information may be derived based on the fusion of several measurements that come from different homogeneous or heterogeneous sensors. For example, in order to determine if an object is placed on a table requires monitoring the output of table's position and weight sensors.

Having acquired the necessary context we are in a position to assess an artifact state (Reasoning Level) and decide appropriate response activation (Planning Level). Adopting the definition from Artificial Intelligence, a state is a logical proposition defined over a set of context measurements (Russell and Norvig, 2003). This state assessment is based on a set of rules defined by the artifact developer. The reaction may be as simple as turn on an mp3 player or send an SMS to the user, or it may be a complex one such as the request of a specific service, e.g. a light service. Such a decision may be based on local context or may require context from external sources as well, e.g. environmental context, location, time, other artifacts. The low (sensor) and high (fused) level data, their interpretation and the local and global decision-making rules are encoded in the application ontology. The basic goal of this ontology is to support a context management process that is based on a set of rules which determine the way that a decision is taken and must be applied on existing knowledge represented by this ontology. The description of the different types of these rules is given in the next section.

20.4.2 Rules

The application model specifies the behavior of an application and in our case this behavior is represented by Event-Condition-Action (ECA) rules. The categories of rules that will support the decision-making process in the context management framework are as follows.

20.4.2.1 Rules for Artifact State Assessment

The left part of these rules denotes the parameters that affect the state of an artifact and the thresholds or the values for these specific parameters that lead to the activation of the rule, while the right part of these rules denotes the artifact state that is activated. These rules support the "translation" of low level context (values of parameters measured by sensors) to state assessment; they may also incorporate the translation from low level context to high level context (e.g. perform a set of operations to values measured by sensors like estimate the average value).

20.4.2.2 Rules for the Local Decision-Making Process

These rules exploit exclusively knowledge from the artifact that uses them. Their left part denotes the artifact states that must be detected and their possible activation level and their right part denotes the artifact requests and needs. When an artifact has a specific need we can consider that it needs a type of service offered by another artifact. When a rule from this category is activated, the artifact has to search its synapses in order to find a synapse which is associated to another artifact plug that provides the requested service. If such a synapse is found then the artifact can select it in order to satisfy its need. The situations, where more than one synapse is found that may be used to satisfy the request or no synapses are found, are handled by the local decision process using another kind of rules. The rules that define the final reaction of an artifact can be defined by the user or can be based on specifically user-defined policies. These rules support both the context delivery and the reaction of an artifact based on the local decision from state assessments.

20.4.2.3 Rules for the Global Decision-Making Process

These rules are similar to the rules for the local decision-making. Their main difference is that the rules for the global decision-making process have to take into account the states of other artifacts and their possible reactions so that to preserve a global state defined by the user.

20.4.3 Implementation

The architecture of the system that implements the aforementioned context management and reasoning process is shown in Fig. 20.3. The core modules of this system, Ontology Manager, Rule Manager and Inference Engine, are part of the updated version of the GAS-OS kernel (Drossos et al., 2007).

The Ontology Manager is the module responsible for the manipulation of knowledge represented into the artifact ontology. Specifically, it can only query the artifact

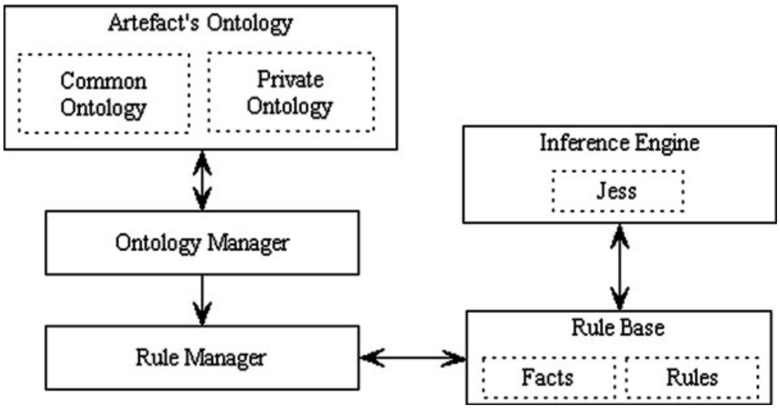

Fig. 20.3 Systems' architecture

common (core) ontology, since this ontology cannot be changed during the deployment of an application. On the other hand, it can both query and update the artifact private (application) ontology. The basic functionality of the Ontology Manager is to provide the other modules of the system with knowledge stored in the artifact ontology by adding a level of abstraction between them and the ontology.

The Rule Manager manages the artifact rule base and is responsible for both querying and updating this rule base. Note that the rules stored in an artifacts' rule base may only contain parameters, states and structural properties that are defined into the artifacts' private ontology. For the initialization of the context management process, apart from the rules, a set of initial facts are necessary. The Rule Manager is also responsible for the creation of a file containing the initial facts for an artifact. For example an initial fact may define the existence of an artifact by denoting its parameters, states and reactions that can participate in its rules and their initial values. In order to create such an initial fact, the Rule Manager uses knowledge stored in the artifacts' ontology. Subsequently, it queries the Ontology Manager for any information that it needs, like the artifacts' parameters, states and reactions.

The Inference Engine supports the decision-making process and is based on the Jess rule engine (Java Expert System Shell) (Jess, 2007). In order to initialize its process execution, the Inference Engine needs the artifact initial facts, which are defined by the Rule Manager, and the rules stored in the rule base. Note that in the current version of our system the rules in the rule base are stored in Jess format. The Inference Engine is informed of all the changes of parameters values measured by artifacts sensors. When it is informed of such a change, it runs all the rules of the rule base. If a rule is activated, this module informs the artifacts operating system of the activation of this rule and the knowledge that is inferred. The artifact's state and reaction is determined from this inferred knowledge.

The ontology describes the semantics of the basic terms of our model for Ubiquitous computing applications and their interrelations. One of the ontology goals is to describe the services that artifacts provide in order to support a service discovery mechanism. Thus the Ontology Manager provides methods that query the ontology for the services that an artifact offers as well as for artifacts that provide specific services. GAS-OS gets from the Ontology Manager the necessary service descriptions stored in the artifact local ontology, in order to implement a service discovery mechanism. Finally the Ontology Manager using this mechanism and a service classification can identify artifacts that offer semantically similar services and propose objects that can replace damaged ones. Therefore, it supports the deployment of adaptive and fault-tolerant Ubiquitous computing applications.

20.4.4 Engineering Applications

To achieve the desired collective functionality a user has to form synapses by associating compatible plugs, thus composing applications using eEntities as components. The idea of building Ubiquitous computing applications out of components is feasible only in the context of a supporting component framework that acts as a middleware. The kernel of such a middleware is designed to support basic functionality such as accepting and dispatching messages, managing local hardware resources (sensors/actuators), the plug/synapse interoperability and a semantic service discovery protocol.

In terms of the application developer, plugs can be considered as context-providers that offer high-level abstractions for accessing context (e.g. location, state, activity, etc.). For example, an *eLamp* may have a plug that outputs whether the *eLamp* is switched on or switched off and an *eRoom* a plug informing if someone is in this room or not. In terms of the service infrastructure (middleware), they comprise reusable building blocks for context rendering that can be used or ignored depending on the application needs. Each context-provider component reads input sensor data related to the specific application and can output either low level context information such as location, time, light level, temperature, proximity, motion, blood pressure or high-level context information such as activity, environment and mood. An artifact has two different levels of context; the low level which contains information acquired from its own sensors and the high level that is an interpretation of its low level context information based on its own experience and use. Additionally an artifact can get context information from the plugs of other artifacts; this context can be considered as information coming from a "third-person experience".

The application developers may establish synapses between plugs to denote both their preferences and needs and to define the behavior of the Ubiquitous computing application. From the service infrastructure perspective, the synapses determine the context of operation for each artifact; thus each artifact's functionality is adapted to the Ubiquitous computing application's structure.

By providing users with this conceptual model, we manage to decouple the low-level context management from the application business logic, which is captured as expressions in terms of high-level concepts that are matched with services available in the current application context. Instead of the classical approach of using established interfaces for resource access, this approach decouples the high-level concepts from the instances implemented by each context.

20.5 Prototype Application Example

20.5.1 Scenario

The example illustrated in this section deals with establishing communication between plants and artifacts. The prototype is a Ubiquitous computing application that is deployed in an indoor environment (e.g. home, office) and aims at facilitating the user in looking after her indoor plants. This ambient intelligence scenario demonstrates the concept of "communicating plant" (Goumopoulos et al., 2004) in the context of an every-day environment with several layers of decision-making.

The scenario is quite simple. A person has a plant in her office. However busy she may be, she still loves to take care of the plant. Several everyday objects are at her disposal for giving her an appropriate feedback (Lamp, MP3Player, Mobile Phone). Our aim is to form such an application where the plant will be able to provide the human with the appropriate feedback about its condition. The sequence of the scenario's interactions is shown in Fig. 20.4.

The core artifact is the *ePlant*, which is constructed by adding sensors to the soil or the leaves of a plant. The *ePlant* "decides" whether it needs water or not by using its sensor readings and the appropriate decision making mechanism incorporated in it. The *eCarpet* is used to record whether the plant owner is inside her office or has stepped outside. Similarly, a *eMoodCube* (a glass brick that the user can set in

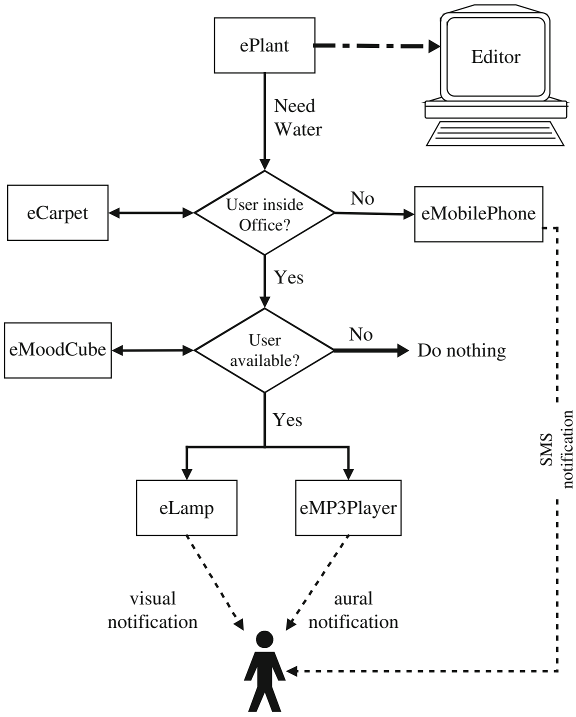

Fig. 20.4 Smart plant flowchart diagram

one of six possible positions) is used to provide an indication whether the user is available or not.

An augmented Lamp (*eLamp*) and an MP3Player (*eMP3Player*) are used to provide visual and aural notification respectively to the user. This notification is given according to the status of the *eMoodCube*. If the *eMoodCube* is set to "Do not disturb" status (which has been mapped by the plant owner to one of the six possible positions), the user is not notified at all. Lastly, in the case the user is not in her office, the application uses the *eMobilePhone* (an augmented Mobile Phone) to send her a SMS and inform her about the watering needs of the plant.

20.5.2 Components

The artifacts that are necessary for the realization of the above scenario are described as follows:

ePlant: The *ePlant* is capable of deciding whether it needs water or not, based on its sensor readings. These sensors fall into two categories: *Thermistors*, that is sensors that can perceive the temperature of the plants leaves and the environment and *Soil Moisture Probes*, which are able to measure the moisture level of the soil. Decision making rules, specific to the plant species, which combine the information given from the sensors above in order to provide a concrete decision on the current plant's state, are added in the *ePlant* local ontology.

eMobilePhone: The *eMobilePhone* is a personal java enabled mobile phone, used for sending SMS to other mobile phones. When it receives a request to notify the user via an SMS, it will send the SMS to the corresponding telephone number.

eLamp: The *eLamp* is an augmented floor lamp used for visual notifications. The lamp switch is controlled by a micro-controller, which adjusts the state of the *eLamp* accordingly.

eCarpet: The *eCarpet* is an augmented carpet with pressure sensors attached, used for sensing the occupancy of the office (i.e. if the user is in the office). Based on the sequence that the pressure sensors are pressed, the *eCarpet* is capable of deducing if someone is entering or leaving a room, thus if the user is present or not.

eMoodCube: The *eMoodCube* is a glass brick containing a set of colored light bulbs with tilt switches attached. Each of the six different positions can be used for defining the current status or mood of the user.

eMP3Player: The *eMP3Player* is used to play audio files.

20.5.3 Implementation

A high-level view of the plugs and synapses that are necessary for the implementation of the Smart Plant application is given in Fig. 20.5.

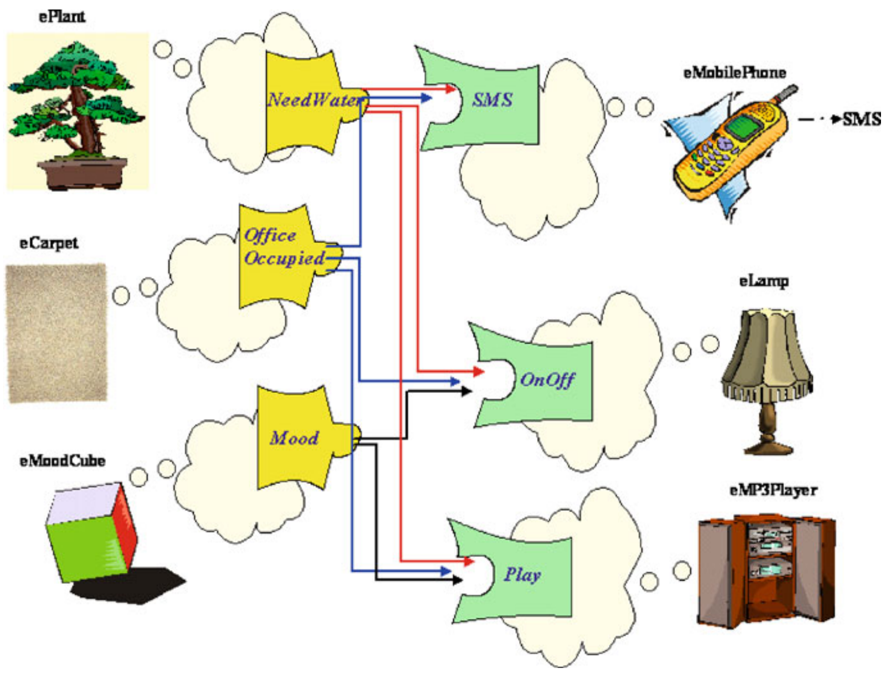

Fig. 20.5 Plugs and synapses for the implementation of the smart plant application

We note that for the *ePlant*, *eCarpet* and *eMoodCube,* the evaluation of the state, service or capability is based on a *local decision making scheme*, because the assessment logic depends only on knowledge that is managed by the specific component through its attached sensor network and rule-base. On the other hand, the service provided by the *eMobilePhone*, *eLamp* and *eMP3Player* depends on a *global decision making scheme*, because the rules that govern the decision to offer a service have to take into account the state and capability information of several eEntities. For example, to decide whether to make the *eLamp* blink (as a visual notification service to the user), we have to take into account the state of the *ePlant*, provided by the *NeedWater* plug, the capability of the *eCarpet* to sense the presence of the user in the office, provided by the *OfficeOccupied* plug, and the capability of the *eMoodCube* to map the mood of the user, through the *Mood* plug. Thus, to turn on or off the *eLamp* we have to define a rule that takes into account all the above plugs.

The following table summarizes the properties, plugs and operation rules (functional schemas) of each eEntity participating in the Smart Plant application. This knowledge is part of the application ontology defined for the application. We have omitted information, such as physical properties, or other plugs, which may reflect services that are not required by the specific application (Table 20.1).

By specifying the rule structure and semantics in an ontology that defines various parameter and state types, as well as the arguments that the rules are based upon,

Table 20.1 Smart plant application ontology configuration (part of)

eEntity	Properties	Plugs	Functional schemas

ePlant	Determining the state of the plant, whether the plant needs irrigation or not.	NeedWater: {OUT|Boolean}	`PlantTemp ← read(Thermistors)`
			`SoilHumidity ← read(MoistureProbe)`
			`AmbientTemp ← read(Thermistors)`
			`IF PlantTemp - AmbientTemp >`
			`ePlant.TempThreshold OR SoilHumidity <`
			`ePlant.HumidityThreshold`
			`THEN NeedWater ← TRUE`
			`ELSE NeedWater ← FALSE`

eCarpet	The carpet is placed at the entrance of the office. As the user enters or leaves he/she is forced to step over the carpet. The eCarpet monitors the direction of this movement and updates its plug accordingly.	OfficeOccupied: {OUT|Boolean}	`SensorArray ← read(SensorNetwork)`
			`OfficeOccupied ←`
			`FindMovementDirection(SensorArray)`

eMoodCube	As the moodCube is turned it changes its color and represents user's mood. Possible selections are:
	• DO_NOT_DISTURB,
	• NOTIFY_VISUAL,
	• NOTIFY_ACOUSTIC
		Mood: {OUT|Enumeration}	`Position ← read(Sensors)`
			`Mood ← MapPositiontoMood(Position)`

eMobilePhone	Send SMS Service (S1)	SMS: {IN|Boolean}	`IF ePlant.NeedWater AND NOT`
			`eCarpet.OfficeOccupied`
			`THEN S1()`

eLamp	Light service (S2)	OnOff: {IN|Enumeration}	`IF ePlant.NeedWater AND`
			`eCarpet.OfficeOccupied AND eMoodCube.Mood =`
			`NOTIFY_VISUAL`
			`THEN S2(BLINK)`

eMP3Player	Playing message service (S3)	Play: {IN|Boolean}	`IF ePlant.NeedWater AND`
			`eCarpet.OfficeOccupied AND eMoodCube.Mood =`
			`NOTIFY_ACOUSTIC`
			`THEN S3(A user-defined audio message)`

we can use the ontology to verify the validity of the rules. This also facilitates the inclusion of context parameters in rules, since we know the rule structure and the value types of different arguments. Furthermore, the use of ontological descriptions allows heterogeneous entities to interoperate and interact with one another in a way dictated by the application domain under examination.

20.5.4 Semantic-Based Service Discovery

Service and resource discovery will play an integral role in the realization of Ubiquitous computing systems. Since artifacts are resource limited, they must be able to discover and use the services of neighboring devices. A service discovery mechanism is needed so that if a synapse is broken, e.g. because of an artifact failure, another artifact that offers a semantically equivalent service could be found. In the example application scenario discussed previously, suppose that the synapse between *ePlant* and *eLamp* is broken because of *eLamp* failure. Then, a new artifact having a property that provides the service "light" must be found. Therefore, for a Ubiquitous computing environment the infrastructure must be enhanced to provide a semantic-based service discovery, so that it is possible to discover all the relevant services.

Since for the Ubiquitous computing applications a semantic service discovery mechanism is useful and the replacement of artifacts depends on the services that artifacts offer, a service classification is necessary. In order to define such a service classification we first identified some services that various artifacts may offer; for the application scenario discussed indicative services are presented in

Table 20.2. From these it is clear that the services offered by artifacts depend on artifacts physical characteristics and/or capabilities and their attached sensors and actuators.

Next we had to decide how we should classify the services. The classification proposals that we elaborated are the following: by object category, by human senses and based on the signals that artifacts sensors/actuators can perceive/transmit. We decided to combine these proposals so that to describe a more complete classification. So we initially defined the following elementary forms of signals that are used: sonic, optic, thermal, electromagnetic, gravity and kinetic. These concepts are

Table 20.2 Services that may be offered by artifacts

Artifact	Offered services
ePlant	needs water yes/no, needs nutrients yes/no, species, other physical characteristics.
eCarpet	object on it yes/no, objects' position, direction, pressure, weight, frequency
eMoodCube	current position
eMobilePhone	send sms, send email, make phone call, get calendar, get contacts
eLamp	switch on/off, light, heat
eMP3Player	sound, sound volume, kind of music, play/pause/stop, next/previous track

divided into lower level services (subclasses); e.g. the sonic service may be music, speech, environmental sound, and noise. Additionally services may have a set of properties; e.g. sonic can have as properties the volume, the balance, the duration, the tone, etc. Finally we enriched this classification by adding services relevant to environmental information, like humidity and temperature.

We have defined a lightweight Resource Discovery Protocol for eEntities (eRDP) where the term resource is used as a generalization of the term service. The protocol makes use of typed messages codified in XML. Each message contains a header part that corresponds to common control information including local IP address, message sequence number, message acknowledgement number, destination IP address(es) and message type identification. The prototype was written in Java using J2ME CLDC platform. kXML is used for parsing XML messages.

One of the ontology goals is to describe the services that the artifacts provide and assist the service discovery mechanism. In order to support this functionality, the Ontology Manager provides methods that query the application ontology for the services that a plug provides as well as for the plugs that provide a specific service. Therefore, the Ontology Manager provides to the calling process the necessary knowledge (which is retrieved from the artifact) that is relevant to the artifact services, in order to support the service discovery mechanism. Similarly the Ontology Manager can answer queries for plug compatibility and artifact replace-ability.

Let's return to the scenario discussed above, where the synapse between *ePlant* and *eLamp* is broken. We said that when this happens, the system will attempt to find a new artifact having a plug that provides the service "light". The system software is responsible to initiate this process by sending a message for service discovery to the other artifacts that participate in the same application or are present in the surrounding environment. This type of message is predefined and contains the type of the requested service and the service's attributes. A description of the *eLamp* service is shown in Fig. 20.6.

When the system software of an artifact receives a service discovery message, it forwards the message to the Ontology Manager. Let's assume that the artifact *eBunny* is available and that this is the first artifact that gets the message for service discovery. The *eBunny* Ontology Manager firstly queries the application ontology of *eBunny* in order to find if this artifact has a plug that provides the service "light". If

```
<res_spec>
<res name> eLamp </ res name>
<res classification> light </res classification >
<res id> eRDP:PLUG:CTI-RU3-eLamp-ONOFF_PLUG </res id  >
<res location> 150.140.30.5 </res location>
<res data> <attrName="power" type="bool" value="false"
<attrName="luminocity" type="integer", value="10"
</res data>
<res timestamp> 4758693030 </res timestamp>
<res expiry> Never </res expiry>
</res_spec>
```

Fig. 20.6 XML description of eLamp service

we assume that the *eBunny* has the plug "*LampBlink*" that provides the service light, the Ontology Manager will send to the system software a message with the description of this service. If such a service is not provided by the *eBunny*, the Ontology Manager queries the *eBunny* application ontology in order to find if another artifact, with which the *eBunny* has previously collaborated, provides such a service. In case of a positive answer it returns as a reply the description of this service. If the queried artifact, in our example the *eBunny*, has no information about an artifact that provides the requested service, the control is sent back to system software, which is responsible to send the query message for the service discovery to another artifact.

20.6 Conclusions

The ontology and the context management framework that we developed sufficiently supports the composition of context-aware Ubiquitous computing applications from everyday enhanced physical objects and it also address a number of key issues of such applications like application model dynamic adaptability and semantic service discovery. The context model that we selected for both these Ubiquitous computing applications is the same ontology-driven model.

Future Ubiquitous computing environments will involve hundreds of interacting and cooperating devices ranging from unsophisticated sensors to multi-form actuators. Although the majority of these devices may have limited resources (computation, memory, energy, etc) or may be only oriented to certain tasks, their collective behavior that results from local interactions with their environment may cause coherent functional global patterns to emerge. Hence, the combination and cooperation of locally interacting artifacts with computing and effecting capabilities may trigger the continuous formation of new artifact ecologies that provide services not existing initially in the individuals and exhibit them in a consistent and fault-tolerant way. As these societies are dynamically reconfigured aiming at the accomplishment of new or previous related tasks, their formation heavily depends not only on space and time but also on their context of previous local interactions, previous configured teams, successfully achieved goals or possibly failures. This means that in order to initially create, manage, communicate with, and reason about, such kinds of emergent ecologies, we need somehow to model and embed to these entities social memory, enhanced context memory, and shared experiences. One step to this end is the design and implementation of evolving multidimensional ontologies that will include both nonfunctional descriptions, and rules and constraints of application, as well as aspects of dynamic behavior and interactions.

References

Bergman, E. 2000. *Information appliances and beyond.* San Francisco, CA: Morgan Kaufmann Publishers.

Bunge, M. 1977. *Treatise on basic philosophy: Volume 3: Ontology I: The furniture of the world.* Dordrecht: Reidel.

Bunge, M. 1979. *Treatise on basic philosophy: Volume 3: Ontology II: A world of systems*. Dordrecht: Reidel.

Chen, H., T. Finin, and A. Joshi. 2003. An ontology for context aware pervasive computing environments. *Knowledge Engineering Review* 18(3):197–207.

Christopoulou, E., and A. Kameas. 2005. GAS ontology: An ontology for collaboration among ubiquitous computing devices. *International Journal of Human-Computer Studies* 62(5):664–685.

De Bruijn, J. 2003. Using ontologies – enabling knowledge sharing and reuse on the semantic web. Technical Report DERI-2003-10-29. Austria: Digital Enterprise Research Institute (DERI).

De Paoli, F., and M. Loregian. 2006. Context-aware applications with distributed ontologies. In Proceedings of the CAISE*06 Workshop on Ubiquitous Mobile Information and Collaboration Systems (UMICS ′06), CEUR Workshop Proceedings 242, eds. M.C. Norrie, S. Dustdar, and H. Gall, 869–883.

Dey, A.K. 2001. Understanding and using context, personal and ubiquitous computing. *Special Issue on Situated Interaction and Ubiquitous Computing* 5(1):4–7.

Dey, A.K., D. Salber, and G.D. Abowd. 2001. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction Journal* 16(2–4):97–166.

Dearle, A., G.N.C. Kirby, R. Morrison, A. McCarthy, K. Mullen, Y. Yang, R.C.H. Connor, P. Welen, and A. Wilson. 2003. Architectural support for global smart spaces. In Proceedings of the 4th International Conference on Mobile Data Management (MDM'03), 153–164. London: Springer.

Disappearing Computer Initiative. 2007. http://www.disappearing-computer.net/. Accessed on 2 Aug 2007.

Dobson, S., and P. Nixon. 2004. More principled design of pervasive computing systems. In Proceedings of Engineering for Human-Computer Interaction and Design, 292–305. Berlin: Springer.

Drossos, N., C. Goumopoulos, and A. Kameas. 2007. A conceptual model and the supporting middleware for composing ubiquitous computing applications. *Journal of Ubiquitous Computing and Intelligence American Scientific Publishers (ASP)* 1(2):1–13.

Ejigu, D., M. Scuturici, and L. Brunie. 2007. CoCA: A collaborative context-aware service platform for pervasive computing. In Proceedings of the International Conference on Information Technology, IEEE computer society, 297–302.

Goumopoulos, C., E. Christopoulou, N. Drossos, and A. Kameas. 2004. The PLANTS system: Enabling mixed societies of communicating plants and artefacts. In Proceedings of the 2nd European symposium on Ambient intelligence (EUSAI 2004), eds. P. Markopoulos, B. Eggen, E.H.L. Aarts, and J.L. Crowley, 184–195. London: Springer.

Gruber, R. 1993. A translation approach to portable ontology specification. *Knowledge Acquisition* 5(2):199–220.

Guizzardi, G., H. Herre, and G. Wagner. 2002. On the general ontological foundations of conceptual modeling. In Proceedings of the 21st International Conference on Conceptual Modeling, eds. S. Spaccapietra, S.T. March, and Y. Kambayashi, 65–78. London: Springer.

Henricksen, K., J. Indulska, and A. Rakotonirainy. 2002. Modeling context information in pervasive computing systems. In Proceedings of the 1st International Conference on Pervasive Computing (Pervasive 2002), eds. F. Mattern, and M. Naghshineh, 167–180. London: Springer.

Jess, 2007. Rule engine for the java platform. http://herzberg.ca.sandia.gov/jess/. Accessed on 2 Aug 2007.

Kameas, A., S. Bellis, I. Mavrommati, K. Delaney, M. Colley, and A. Pounds-Cornish. 2003. An architecture that treats everyday objects as communicating tangible components. In Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communications (PerCom03), 115–122. IEEE CS Press.

Kagal, L., V. Korolev, H. Chen, A. Joshi, and T. Finin. 2001. Centaurus: A framework for intelligent services in a mobile environment. In Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCSW'01), 195–201. IEEE Computer Society.

Kindberg, T., J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino, B. Serra, and M. Spasojevic. 2000. People, places, things: Web presence for the real world. Technical Report HPL-2000-16, HP Labs.

Milton, S.K., and E. Kazmierczak. 2004. An ontology of data modelling languages: A study using a common-sense realistic ontology. *Journal of Database Management* 15(2):19–38.

Norman, D. 1999. *The invisible computer*. Cambridge, MA: MIT Press.

Ranganathan, A., and R. Campbell. 2003. An infrastructure for context-awareness based on first order logic. *Personal and Ubiquitous Computing* 7(6):353–364.

Rosemann M., and P. Green. 2000. Integrating multi-perspective views into ontological analysis. In Proceedings of the 21st International Conference on Information Systems, 618–627. Brisbane: Association for Information Systems.

Rosemann M., and P. Green, and M. Indulska. 2004. A reference methodology for conducting ontological analyses. In Proceedings of Conceptual Modeling – ER 2004, 110–121. London: Springer.

Russell, S., and P. Norvig. 2003. *Artificial intelligence: A modern approach*, 2nd edition. Upper Saddle River, NJ: Prentice Hall.

Gu, T., X. H. Wang, H .K. Pung, and D.Q. Zhang. 2004. An ontology-based context model in intelligent environments. In Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS'04), 270–275. Society for Computer Simulation.

Strang, T., and L. Linnhoff-Popien. 2004. A context modelling survey. Proceedings of the 1st International Workshop on Advanced Context Modelling, Reasoning and Management. http://www.itee.uq.edu.au/∼pace/cw2004/Paper15.pdf.

Strang, T., L. Linnhoff-Popien, and K. Frank. 2003. CoOL: A context ontology language to enable contextual interoperability. In Proceedings of the 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003), 236–247. Springer.

Wand, Y., and R. Weber. 1990. An ontological model of an information system. *IEEE Transactions on Software Engineering* 16(11):1282–1292.

Wang, X.H., D.Q. Zhang, T. Gu, and H.K. Pung. 2004. Ontology based context modeling and reasoning using OWL. In Proceedings of the 2nd IEEE Annual Conference on Pervasive Computing and Communications Workshops. 18. IEEE Computer Society.

Weber, R. 1997. Ontological foundations of information systems. Coopers & Lybrant Accounting Research Methodology. Monograph No. 4.

Weiser, M. 1991. The computer for the 21st century. *Scientific American* 265(3):94–104.