

# A Programming Model for Pervasive Spaces

Erwin Jansen, Bessam Abdulrazak, Hen-I Yang, Jeff King, and Sumi Helal  
Computer & Information Science & Engineering University of Florida Gainesville,  
FL-32611

**Abstract.** In this article we will propose a formal model for smart spaces. The key ingredients of this model are user, sensors and actuators. We show how we can derive a programming model based upon knowledge and software engineering. We describe how to interpret the world using an ontology and use this ontology to describe the intentional effect of available actuators. The model is loosely based on the belief-desire-intention model and is a significant simplification of other context aware middleware architectures. We demonstrate how we can implement this model using OSGi.

## 1 Introduction

The last few years we have seen an increased interest in the area of ubiquitous computing. The idea behind ubiquitous computing is that the technology should be calm [1], in the sense that many computers are available throughout the physical environment, while making them effectively invisible to the user [2]. Hence ubiquitous computing is sometimes referred to as pervasive computing. It is considered to be the third wave of computing.

We consider a smart space to be an ambient [3], in the sense that there is a boundary that determines what is part of the smart space, and what is not. Inside the smart space computations take place, to assist the users of the space. A pervasive space consists of a collection of devices and software that controls these devices. There are devices that sense the environment (called sensors) and devices that can change the environment (called actuators). The key to a pervasive space is that it is there to assist a user to accomplish his or her task.

This gives rise to a fundamentally different form of interaction. In traditional computer systems, or non-pervasive space for that matter, a user is always telling the system what should happen next. The user is always dictating what the next course of action should be. In a pervasive space we try to make this more fluent, in the sense that the space tries to anticipate the preference of the user and act upon it.

In this article we propose a programming model based upon ontologies and behavior. A software engineer defines an ontology describing the smart space. The concepts interpret possible values that come from a sensor. For example we can associate the concept light with certain output ranges of a light sensor. Next the programmer associates behavior with the concepts that have been defined.

We use this ontology to describe the *intentional* effect of the various actuators. For example we can describe the concepts light and dark and use these concepts to express that a lamp when turned on will lead to the context light.

This description allows us to reason about the behavior of an actuator. We can for example determine that simultaneous invocation of a heater and air-conditioning might not be beneficial. This way we can catch programming mistakes at compile time.

We first start by looking at related work, after this we will identify which parts of a pervasive space are of interest. We will then proceed by formalizing actuators, sensors and the interpretation of the sensors. Based upon our formalization we will present the denotational semantics of a pervasive space and demonstrate how we can implement this model using OSGi.

## 2 Related Work

Most research projects that involved ubiquitous computing have been pilot projects that show that pervasive computing is usable[4]. These pilot projects represent ad hoc specific solutions that are not easy to replicate. At the heart of these applications is the notion of context.

Context aware computing is a computing paradigm in which applications can discover and take advantage of contextual information. This could be temperature, location of the user, activity of the user, etc. Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves[5].

In order to ease the development of pervasive application effort has been placed into developing solutions that enable easy use of context. There are two main approaches: libraries and infrastructure. A library is a generalized set of related algorithms whereas an infrastructure is a well-established, pervasive, reliable, and publicly accessible set of technologies that act as a foundation for other systems. For a comparison between the two see [6].

The context toolkit [7, 8] provides a set of java objects that addressed the distinction between context and user input. The context consists of three abstractions: widgets, aggregators and interpreters. Context widgets encapsulate information about a single piece of context, aggregators combine a set of widgets together to provide higher level “widgets” and interpreters interpret both of these.

Related to this work is the notion of the semantic web services[9]. The idea is that by giving a description of a webservice we can automate webservice tasks like automated discovery, invocation, composition and interoperation. The semantic web makes use of a description logic to describe a webservice. Description logics are knowledge representation languages tailored for expressing knowledge about concepts and concept hierarchies. Using this description an agent can reason about the behavior or effect of the invocation of a webservice.

The SOCAM architecture [10] is a middleware layer that makes use of ontologies and predicates. There is an ontology describing the domain of interest. By making use of rule based reasoning we can then create a set of rules to infer the status of a an entity of interest. Their research shows that ontologies are usable but require some computation power and hence are not suitable for smaller devices.

The CoBra architecticture [11] is a broker center agent architecture. At the core is a context broker that builds and updates a shared context model that is made available to appropriate agents and services.

The framework presented here differs in the fact that we give a rigid formalization and solely make use of an ontology for classification and activation of behavior. The advantage of this approach is that we can use the established ontology to describe the *intentional* effect of actuators. This in turn allows us to reason about the behavior of the various actuators. We can detect conflicting behaviors, as well as circular behaviors, and inform the programmer.

### 3 Formalizing a Space

A smart space consists of three entities that are of relevance:

**User:** A user is part of the space. A user interacts with the space and has a set of preferences and desires. A user is in a particular context (or state). Based on the context a user has a different set of desires about the behavior of the space. For example if the user is sleeping, he has no desire for a stereo at a loud volume.

**Sensors:** Sensors sense a certain value in a particular domain. They provide information to the system about the current state of the space. Sensor cannot change the state of the space, they can only observe. A temperature sensor for example can only observe the current temperature, not influence it.

**Actuators:** Actuators influence the state of the space. The invocation of an actuator has a so called *intentional* effect on a particular domain.. This intentional effect might be observable by a sensor. The intentional effect of a heater is to raise the temperature, which can be observed by a temperature sensor.

The aim of a truly smart space is to identify the desires and intentions of its users and act upon these desires with a minimal amount of involvement of the user. For example, a smart space could realize that the favorite television program is playing while the user is sleeping. After sensing this situation the space could automatically record the program.

### 4 The Physical World

Smart spaces deal with the actual, physical world. We observe and interact with the world. We consider the world to be in a certain state at a given time. For

the sake of simplicity we will consider our universe of interest to be  $U = \prod D_j$  where  $D_j$  is an *observable* domain of interest that lies within the bounds of the space. With observable we mean that there exists a sensor (somewhere) that is capable of observing that domain, although that sensor might not be present in the space.

We will use  $u \in U$  denote the current state of the world. We will use  $u_j$  to denote the value of at position  $j$  in the tuple, hence  $u_j \in D_j$ .

In the smart space we have a set of physical devices with which we can interact. In this model we only consider devices, called actuators, that we can turn on or off. An actuator is capable of changing the state of the world. Sensors, in turn, may observe the effect of an actuator. For example a light sensor may observe that the smart house or the resident turned on a lamp.

We model the actuators as a set  $A$  with typical elements  $a_i \in A$ . We denote  $A_{on} \subset A$  to be the set of actuators that are on and  $A_{off} \subset A$  the actuators that are turned off. An actuator is either on or off so  $A_{on} \cap A_{off} = \emptyset$ . Throughout this paper we assume that an actuator that is turned off has no effect on the world. From the mathematical perspective an actuator that is off is the same as an actuator that is not in the room.

#### 4.1 Denotational Semantics

Using actuators only we can give a simple model that describes the effect of invoking a set of actuators. In order to do this we introduce a function that describes how the *world* evolves given a set of actuators that are turned on:

**Definition 1 (Actuator Effect)** *The effect of a invoking a set of actuators is:*

$$G : U \times A_{on} \rightarrow U$$

Basically the function  $G$  captures the behavior of the world. In reality we don't have access to the actual function  $G$ , but we can observe the effect of the function  $G$ . The function  $G$  describes for example how the amount of light changes in the space when we turn on a lamp.

Based upon the world and actuators we can establish an extremely simple model for computations with a set of actuators. In our language we only allow actuators to be turned on and off.

**Definition 2 (Basic actions)** *Given a set of actuators  $A$  the basic actions of the programming language are defined as follows:*

$$act ::= \uparrow a_i \mid \downarrow a_i$$

The semantics of the language constructs are as follows:  $\uparrow a_i$  turns the actuator  $a_i \in A$  on.  $\downarrow a_i$  turns the actuator  $a_i$  off.

These basic actions only make sense if they are combined in a useful manner. We consider an action part of a set of statements that define the flow of control of a smart space.

**Definition 3** Where  $S$  are the statements describing the flow of control:

$$S ::= act \mid S_1; S_2$$

Let  $;$  denote action prefixing. We first perform action  $S_1$  after which we execute the remaining statements  $S_2$ .

**Definition 4 (Simple Programmable Pervasive Space)** A programmable pervasive space is a tuple consisting of:

$$P ::= \langle S, U, A \rangle$$

where  $S$  is the flow of control,  $U$  is the state of the world and  $A$  denotes the set of actuators that are activated.

**Definition 5 (Transition rules)** The transition rules for a pervasive space are now:

1. Transition due to nature:  $\langle S, u, a \rangle \xrightarrow{\tau} \langle S, G(u, a), a \rangle$ .
2. Activation:  $\langle \uparrow a_i, u, a \rangle \rightarrow \langle \epsilon, u, a \cup a_i \rangle$
3. De-activation:  $\langle \downarrow a_i, u, a \rangle \rightarrow \langle \epsilon, u, a \setminus a_i \rangle$

Notice that in this model  $S_1; S_2$  can be executed concurrently, but that  $S_1$  will always be executed before  $S_2$ . We can now define a run of the system as:

**Definition 6 (Run of a System)** Let  $P \xrightarrow{t} P'$  to denote that we can reach  $P'$  via transition  $t$ . Let  $\rightarrow^*$  be the reflexive and transitive closure of  $\rightarrow$ . When  $P = P^0$ ,  $P'$  is said to be run resulting in  $P'$ . We say that  $P'$  is run of  $l$  steps if the sequence of transitions  $\sigma = t_1 t_2 \dots t_l$  is such that  $P \xrightarrow{\sigma^*} P'$ . Where  $P \xrightarrow{\sigma^*} P' = P \xrightarrow{t_1} P^1 \dots \xrightarrow{t_l} P'$ .

## 5 Obtaining Information from Sensors

The denotational semantics given in the previous section do not take into account that we can also observe that state of the space and that we can decide to invoke behavior depending on the observed state of the space.

Sensors provide information about the current state of affairs. At the lowest level we have hardware sensors that produce a constant stream of output values in a particular domain.

**Definition 7 (Base Sensor)** A base sensor is a function that produces an output at a given time in a particular domain.

$$f_i : U \rightarrow D_i$$

A base sensor will *always* be a physical device that can be found in the smart space. Notice that we can have multiple sensors operating in a similar domain. Multiple sensors will be indicated by  $f_i^1, f_i^2$ , etc. It does not necessarily have to be the case that two sensors provide the same value.

We will group all available sensors together and define  $f : U \rightarrow U$  as  $f = \prod f_i$  to be a snapshot of all sensors at that point in time.

It is important to note that our sensors give us a sense of the current state of the world, which does not necessarily have to be correct. Many sensors are incorrect, and some theories even state that it is impossible to properly measure the state of the world without changing it. So  $u_j \neq f_i$ .

## 6 Interpreting Sensor Information

Dealing directly with sensor data is rather awkward. Instead we would like to work with higher level information that describes the state of the world. A natural choice for describing knowledge are description logics. Using a description logic we can define an ontology that describes the smart space. We will restrict ourselves to a taxonomic structures, and therefore will have no need to define roles. The language  $\mathcal{L}$  we define here is similar to  $\mathcal{ALC}$  [12] without roles. For an in depth discussion on description logics see [12]. We will closely follow their definitions:

**Definition 8 (Taxonomy)** *Concept descriptions in  $\mathcal{L}$  are formed using the following syntax rule:*

$$\begin{array}{ll}
 C, D \longrightarrow AC & | \text{ (atomic concept)} \\
 \top & | \text{ (universal concept)} \\
 \perp & | \text{ (bottom concept)} \\
 \neg C & | \text{ (concept negation)} \\
 C \sqcap D & | \text{ (intersection)} \\
 C \sqcup D & \text{ (union)}
 \end{array}$$

We could for example define the atomic concepts *WARM, MODERATE, COLD, DOOROPEN* a derived concept could be something as  $UNWISE = COLD \sqcap DOOROPEN$ .

Apart from describing concepts we need to have a way to interpret these concepts, which can be defined as follows:

**Definition 9 (Interpretation Function)** *We define the interpretation  $\mathcal{I}$  as follows:*

$$\begin{aligned}
AC^{\mathcal{I}} &\subseteq U \\
\top^{\mathcal{I}} &= U \\
\perp^{\mathcal{I}} &= \emptyset \\
(\neg C)^{\mathcal{I}} &= U - C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}}
\end{aligned}$$

The notation  $\mathcal{L}(\mathcal{C})$  is used to denote the set of complex concepts in  $\mathcal{L}$  over the set of atomic concepts  $\mathcal{C}$ . Apart from defining concepts we will make use of the standard ontology definitions:

**Subsumption:**  $C \sqsubseteq D \Leftrightarrow C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$

**Equality:**  $C = D \Leftrightarrow C^{\mathcal{I}} = D^{\mathcal{I}}$

**Membership:**  $C(a) \Leftrightarrow a^{\mathcal{I}} \in C^{\mathcal{I}}$

We say that  $\mathcal{I}$  is a model of a set of statements, if  $\mathcal{I}$  satisfies every statement in  $\mathcal{T}$ . We say that  $\mathcal{T}$  entails a statement  $\gamma$ , i.e.  $\mathcal{T} \models \gamma$ , if and only if every model of  $\mathcal{T}$  also satisfies  $\gamma$ . We can now define an ontology as:

**Definition 10 (Ontology)** *An ontology  $O$  is defined as a tuple:*

$$O = \langle T, \mathcal{C} \rangle$$

Where  $\mathcal{C}$  is the set of atomic concepts and  $T$  is a TBox consisting of a set of descriptions.

It is assumed that the definitions in  $T$  are unique and acyclic. The ontology  $O$  can now be used to interpret the state of affairs in the smart space. We consider the concepts defined in our ontology to be equivalent to the notion of context in the pervasive computing community [13, 14]. We will consider context to be defined in terms of concepts, and hence will use the term concept and context interchangeably.

Interpretation of the current state of the world is straightforward. We identify whether the current state of the world is a member of any concept. I.e. to verify whether  $u \in U$  satisfies concept  $C$  we check that  $u^{\mathcal{I}} \in C^{\mathcal{I}}$ .

The concepts defined earlier can now be given an interpretation:  $COLD = \{0, ..10\}$ ,  $MODERATE = \{11, ..17\}$ ,  $WARM = \{18, ..25\}$ ,  $DOOROPEN = \{doorSensor = \top\}$ .

**Definition 11 (Active Context:)** *The active context of the space  $R : U \rightarrow \mathbf{C}$  is:*

$$R = \{C \mid u^{\mathcal{I}} \in C^{\mathcal{I}}\}$$

Many other researchers use a different method for deriving higher level information. Most sensor networks in the literature make use of a hierarchy of interpretation functions, often referred to as context derivation [15, ?,?].

Context derivation typically is done by making use of derived interpretation functions. These functions take as input the value (or history of values) from other sensors and produce a new output value. The reason why we steer away from these derivations is that derived functions do not have to follow the consistency rules as specified by a description logic. We could easily create a derived sensor that derives an illogical concept. This defeats the usage of a description logic.

## 7 User

So far we have described a space consisting of sensors and actuators. The key element that is missing is the user. To model the user we roughly follow the belief-desire-intention [16] model, in the sense that the user observes the state of the space through its sensors and based upon this information commits itself to a plan of action. We have the following ingredients:

**Desires:** A user has a set of preferences about the state of the world. These preferences dictate how the world should be at that given moment. When a user is going to bed, the user would like the doors to be locked etc.

**Belief/Context:** Is the current observed state of the world, interpreted by the ontology. In any given context the user has a set of preference about the world. When the context changes, the set of preferences change as well.

**Intention:** This is the plan the user commits itself to. In a smart space this would be the activation of a set of actuators. The aim of the activation is to fulfill the desire mentioned before.

The idea is that the space observes that state of the world through the sensors and derive which contexts satisfy the current state. Associated with each context is a set of behaviors. The set of behaviors intend to change the current context to a more desired context. For example, in the context dark we actuate a lamp with the intention to reach the desired context of light. We extend our formal model as follows:

**Definition 12 (Programmable Pervasive Space)** *A programmable pervasive space is a tuple consisting of:*

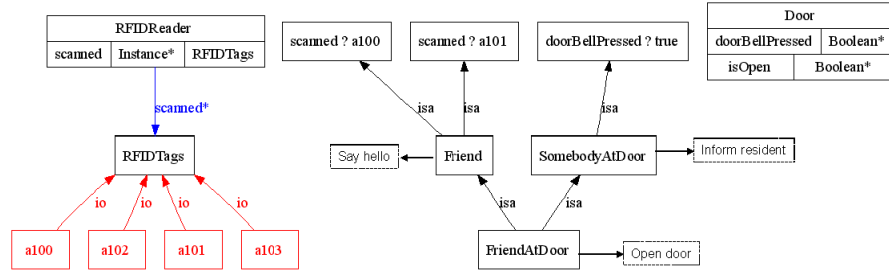
$$P ::= \langle O, F, I, S, U, A_{on} \rangle$$

where  $O$  is the ontology,  $F \subseteq U$  is the observed state of the world through the sensors.  $I : C \rightarrow S$  is a mapping from context to statements. We will write  $\uparrow a_i \in C$  to denote that  $\uparrow a_i$  is associated with context  $C$ , similarly we will write  $\downarrow a_i \in C$  to denote that  $\downarrow a_i$  is associated with  $C$ .

All we need to do is activate behavior whenever the active context changes. Upon a change of context, we activate a new set of actuators.

**Definition 13 (Context change)** *Whenever the context changes, we activate a new set of behaviors.*

$$\langle O, f, i, S, u, A \rangle \rightarrow \langle O, f', i, i(R(f')), f', u, A \rangle \quad \text{if } R(f') \neq R(f)$$



**Fig. 1.** Example of an ontology for a front-door application

A simple example of this framework is the front door application as used in the GatorTech Smarthouse[17]. In figure 1 we have 3 sensors: a doorbell, an RFID reader and a sensor that detects that the door is open. We assume that a person is wearing an RFID tag that we scan upon coming in proximity of the door. The house can now classify whether a friend or an unknown person is at the door. Based upon the classification the appropriate action is undertaken. If a friend is at the door the door is opened, otherwise the owner is informed that there is somebody at the door. Of course, this assumes the user's current desire is for friends to enter the house immediately.

### 7.1 Inheritance of behavior

The function  $i$  defines how we select behavior that we activate upon a context change. A straightforward mapping from all the elements  $R$  to statements leads to a problem with subsumption. If  $C \sqsubset D$  and  $C$  is in the active context it automatically follows that  $D$  is in the active context as well. This implies that all the statements associated with  $C$  as well as those with  $D$  will be activated. This might be undesirable, since  $C$  is more specific than  $D$ . Consider figure 1 and suppose there is a friend at the door. This leads to 3 active contexts, and hence the activation of 3 behaviors.

We solve this problem by overriding behavior. Normally we invoke all the actuators in every active context, however an actuator can be overridden by a more specific context:

**Definition 14 (Overriding behavior)** We say an action  $\downarrow a_i$  overrides  $\uparrow a_i$  (or vice versa) whenever  $C$  is in the active context  $\downarrow a_i \in C$ ,  $C \sqsubset D$  and  $\uparrow a_i \in D$ . Any action that is overridden will not be executed.

The rule of overriding behavior in the frontdoor application assures us that when a friend is at the door only one actuator will be invoked.

## 8 Describing Actuators

Every actuator in the house has a certain intentional effect: the goal we intend to achieve with activation of the actuator. Ideally we can observe the effect of an actuator. If we turn on an actuator it should somehow be observed by our sensors. An actuator has an effect on a domain, which in turn can be observed by a sensor that senses that particular domain. For example, the intentional effect of turning on the heater is to increase the temperature.

We formalize the intentional effect of an actuator as follows:

**Definition 15 (Intentional Effect)** We define the intentional effect of an actuator  $a \in A$  as follows:

$$E \subseteq A \times C \rightarrow C$$

We will write  $a_i : C_1 \rightarrow C_2$  as a shorthand for  $E(a_i, C_1)$ . The meaning of  $a_i : C_1 \rightarrow C_2$  is given the context  $C_1$  the invocation of  $a_i$  will lead to context  $C_2$ .

Using the description of an intentional effect it is now straightforward to identify whether or not two actuators are conflicting with each other.

**Theorem 1.** Two actuators  $a_i$  and  $a_j$  are in conflict in context  $C$  if  $E(a_i, C) \sqcap E(a_j, C) = \emptyset$

*Proof.* This is trivial. The invocation of  $a_i$  leads to a context that is disjoint from the context to which the invocation of  $a_j$  will lead, hence this context can never arise.

A classical example of the invocation of two opposing actuators are the air-conditioning and the heater. Consider the following descriptions:

Heater	Air-conditioning
$COOL \rightarrow MODERATE$	$MODERATE \rightarrow COOL$
$MODERATE \rightarrow WARM$	$WARM \rightarrow MODERATE$

If we invoke the heater and actuator in the context moderate we would end up in a context that is not satisfiable.

The description of an actuator makes it possible for a computer to reason on how to arrive in a particular context. We can construct a directed graph based upon the context definitions and actuator descriptions. The vertices of this graph are the various contexts and the edges are the intentional descriptions of the actuators.

This assists the programmer in the following ways:

**Cyclic Behavior:** From the context to behavior mapping we can derive in which context an actuator will be activated. Since we know to which state an actuator will lead to, we can detect possible cycles. These cycles might be undesirable. For example, if we activate the heater in the context moderate and the air-conditioning in context warm we have a constant invocation of both actuators.

**Actuator Derivation:** If a programmer knows which context is desirable given a particular context, the computer can derive which sequence of actuators need to be activated in order to arrive in that context. For example the programmer can determine that the context moderate is not desirable, but the context warm is. The computer can now derive that it should invoke the heater.

## 9 Implementation

Based upon the framework given in the previous sections the implementation is straightforward. We make use of the OSGi framework. The OSGi framework [18] adds the capability to manage the life cycle of the software components from anywhere in the network. Software components, called bundles, can be installed, updated, or removed on the fly without having to disrupt the operation of the device. Every device in the smartspace will be represented by a bundle. We use OWL-Lite as the description logic to interpret the available sensors.

Every sensor that is part of the smart space will have its own bundle associated with it. This bundle will contain the necessary drivers to obtain information from the sensor as well as an ontological description of this device. The sensor itself is capable of producing raw data. Upon activation the bundle will register the sensor as a service that produces these values.

Attached to every sensor is a context interpreter that interprets the sensor information. The context interpreter maps the sensed value into an atomic concept. If the sensed value gives rise to a new atomic concept, the context manager is informed of this change. The context manager will now reclassify the current state of the world. If this classification leads to a new set of active context the manager will start and stop the bundles associated with these contexts.

## 10 Conclusion and Future Work

We have shown a model that is loosely based upon the BDI model. We program the space by describing an ontology that interprets the domain of all available sensors. We use this ontology to express the effects of available actuators that can actively change the state of the space. We can now program the space by associating behavior with various context states.

The description of the actuators allows us to reason about the effect of the invocation of a particular actuator. This allows us to detect conflicting invocations, cyclic behavior and gives suggestions to a programmer on how to reach a particular context state.

Currently the description of an actuator needs to be given by the programmer. This can be quite cumbersome and is not always feasible. The information that we really need is the ability to predict the outcome of the function  $G$  given the set of active actuators. We are investigating if it is possible to employ learning techniques so we can associate a  $g_{a_i}$  with each actuator  $a_i$  that will predict the outcome of  $G$  when  $a_i$  activated.

Another area of interest is to fully adopt the BDI model. We are considering annotating the various context states with a utility function that describes how desirable the particular context is. The system could then use a shortest path algorithm to see if there exists a path to a more desirable context and invoke the appropriate actuators.

We envision an integrated development environment (IDE) that allows us to visually represent the ontology and the associated behavior. This has been partially implemented already. The simplicity of this model will hopefully allow non-computer scientists to easily program parts of the smart space.

## References

1. Weiser, M., Brown, J.: Designing calm technology. *PowerGrid Journal* **1** (1996)
2. Lyytinen, K., Yoo, Y.: Issues and challenges in ubiquitous computing. *Communications of the ACM* **45** (2002) 62–65
3. Cardelli, L., Gordon, A.D.: Mobile ambients. In: *Foundations of Software Science and Computation Structures: First International Conference, FOSSACS '98*, Springer-Verlag, Berlin Germany (1998)
4. Chen, G., Kotz, D.: A survey of context-aware mobile computing research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College (2000)
5. Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M., Steggle, P.: Towards a better understanding of context and context-awareness. In: *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, Springer-Verlag (1999) 304–307
6. Hong, J., Landay, J.A.: An infrastructure approach to context-aware computing. *Human-Computer Interaction* **16** (2001)
7. Dey, A.K., Salber, D., Abowd, G.D.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction (HCI) Journal* **16** (2001) 97–166
8. Salber, D., Dey, A.K., Abowd, G.D.: The context toolkit: Aiding the development of context-enabled applications. In: *CHI*. (1999) 434–441
9. Coalition, T.O.S.: Owl-s: Semantic markup for web services. <http://www.daml.org/services/owl-s/1.0/owl-s.html> (2004)
10. Gu, T., Pung, H.K., Zhang, D.Q.: A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications (JNCA)* **28** (2005) 1–18
11. Chen, H., Finin, T., Joshi, A., Perich, F., Chakraborty, D., Kagal, L.: Intelligent agents meet the semantic web in smart spaces. *IEEE Internet Computing* **8** (2004)
12. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., eds.: *The Description Logic Handbook*. Cambridge University Press (2002)
13. Chen, G., Kotz, D.: Context aggregation and dissemination in ubiquitous computing systems. In: *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, IEEE Computer Society Press (2002)

14. Meyer, S., Rakotonirainy, A.: A survey of research on context-aware homes. In: Proceedings of the Australasian information security workshop conference on ACSW frontiers. Volume 21., Darlinghurst, Australia, Australia, Australian Computer Society, Inc (2003) 159–168
15. Chen, G., Kotz, D.: Solar: An open platform for context-aware mobile applications. In: an informal companion volume of short papers of the Proceedings of the First International Conference on Pervasive Computing. (2002) 41–47
16. Wooldridge, M.: Reasoning about Rational Agents. The MIT Press, Cambridge, Massachusetts/London, England (2000)
17. Helal, A., Mann, W., Elzabadani, H., King, J., Kaddourah, Y., Jansen, E.: Gator tech smart house: A programmable pervasive space. IEEE Computer magazine (2005) 64–74
18. Alliance, T.O.: OSGi Service Platform, Release 3. Ios Pr Inc (2003)