Formal Modeling in Context Aware Systems

Anjum Shehzad, Hung Q. Ngo, Kim Anh Pham, and S. Y. Lee

Computer Engineering Dept. Kyung Hee University
449-701 Suwon, Republic of Korea
{anjum, nqhung, kimanh, sylee}@oslab.khu.ac.kr

Abstract. Ubiquitous computing environment consists of diverse range of hardware and software entities, and is about the interactivity of such entities. Context-awareness, being an important ingredient plays a vital role in enabling such interactive smart environments. The entities and contextual information provided/utilized by them must have invariant meanings in order to have a common understanding among them. This results in sharing of information with common semantics, at different times and at different places and provides testability of formalized knowledge, emerging as a pool of consistent contextual knowledge available to different context-aware systems. In this paper, we discuss our context model for the home domain and show that how it entails implicit reasoning.

1 Introduction

Ubiquitous computing is viewed as a major paradigm shift from conventional desktop application development. This view is enabled through the use of diverse hardware (sensors, user devices, computing infrastructure etc.) and software, anticipating user needs and acting on their behalf in a proactive manner [1], [2]. This diversity of hardware and software information increases the degree of heterogeneity.

Context-awareness is considered as an important ingredient of today's most ubiquitous computing applications. The behavior of these applications is mostly characterized by embedding the interpretation logic of contextual information inside applications, creating problems for reusability of this information by other applications. Since ubiquitous computing is about interactive and smart environments, in order to enable such interactions, applications need a shared understanding of context to communicate and transfer contextual information effectively among them. Also, the applications demanding the contextual information from the environment may not have its prior knowledge, further emphasizing the need for common agreement of such information. All these problems of heterogeneity, independent interpretation, and need for interactivity leads us to think of a formal context model for efficient utilization of contextual information in ubiquitous computing environment.

Most context-aware systems to date mainly focus on the contents of context, neglecting the importance of interactivity among applications. Some have model the context as name-value pairs [3] and entity relation model, while others have used objects [4] to represent context, with fields containing state of context, methods to access, modify and/or register for notification changes to context. However, context reuse and sharing among wider application domains demand a need for formal context modeling enabling common understanding of the structured context.

The remaining paper is organized as follows. First, we emphasize on formal modeling, its benefits and describe using OWL as formal context modeling language. Next, we explain shortly our architecture, followed by our basic and detailed context model. Finally, reasoning mechanisms, future work and conclusion are provided.

2 Formal Context Modeling

Before we dwell on formal context modeling, we would like to define the meaning of context. The specific conditions, external to the application itself, such as audience, speaker (user), situation (place and its surroundings), time, environmental and network conditions, etc., which determine the application behavior, will be called the 'context' of the application. Such applications which take advantage of the context are called context-dependent or context-aware applications and lead us to the development of context-aware systems [5], [6]. Thus, context-aware systems are more adaptive to such context and more responsive to the user.

Pervasive environments are characterized by different variable entities (context entities). These entities may have different meanings associated with them in different pervasive environments. In order to have invariant meanings of these entities, when used at different times, in different situations, by different applications, they must be formalized, i.e. the context semantics should be formalized. Formalizing the context of an application has a number of clear advantages. First, it allows us to store the context for a long term since its meaning will remain same for future uses. The second advantage is for communicating the context universally with other systems. Third, formal meaning of the context leads to its testability of being a formalized knowledge. So, formalizing the context model helps to make a growing pool of well-tested context knowledge available to different context-aware systems.

2.1 Formal Context Modeling using OWL

Context entities are the concepts in a domain of discourse, and to provide formal meaning of these concepts, ontologies are used, defined as, a formal explicit description of concepts in a domain of discourse, leading to shared and common understanding that can be communicated between people and application systems [7]. Formalizing domain not only contains the vocabularies of concepts but relationships among them as well. W3C's OWL (web ontology language) [8] allows us to achieve this goal in two steps. First, it allows us to define concepts and their inter-relationships e.g. describing person, devices, location etc. Second, it allows us to define instance data pertaining to some specific time and space e.g. Bob is watching television. Traditionally, ontologies are only used to describe domains (as mentioned above) but in OWL, the horizon of ontology has been broadened to include instance data as well, effectively making the knowledge base [7].

OWL, a knowledge representation language, has explicit semantics associated with the knowledge, which provides reasoning capabilities used by intelligent systems and agents to infer useful contexts. As OWL is based on meta-modeling language (RDF [9]), it can be used to represent meta-information about sensors, like in our framework, we are also using OWL to represent access mechanisms to the sensors and associated policies.

3 CAMUS Architecture

The formal context modeling presented in this paper is one part of our CAMUS architecture, a unified middleware framework for context-aware ubiquitous computing. Here we briefly describe the core functional components of CAMUS as depicted in figure 1, more details can be found in another paper [10].

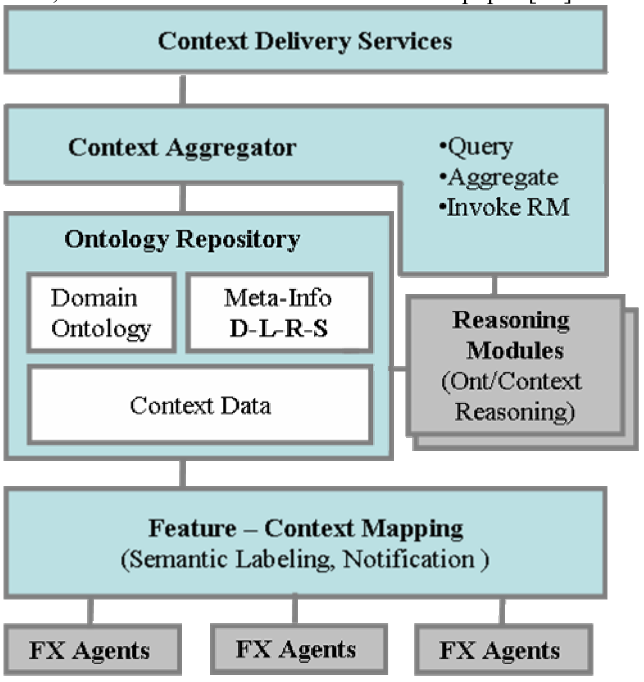

Fig. 1. CAMUS – A Unified Middleware Framework for Context-Aware Ubiquitous Computing. Multi layered abstraction provides separation of concerns and helps in modeling and reasoning contextual information independently from sensing technologies. Context Aggregator is responsible for satisfying certain context queries and providing context to interested applications through Context Delivery Services.

1. Feature Extraction Agents: These sensing agents extract the most descriptive features for deducing contexts in upper layers. In order to have a more expressive representation of contextual information, features are further quantized or segmented, resulting in a set of symbolic values that describe concepts from the real world. The quantized features are encapsulated in the form of Feature Tuple. ***Feature - Context Mapping*** module performs the mapping required to convert a given feature into elementary context based on the meta-information saved in the ontology repository. For example, see Table 1.

2. Ontology Repository: provides the basic storage services in a scalable and reliable fashion and contains the domain ontology (concepts and properties), contextual information (including both elementary and composite contexts), and meta-information (D = devices, S = sensors access mechanisms, L = Feature - Context Labeling, as well as the meta-information about the input, output and capabilities of pluggable reasoning modules = R).

3. Reasoning Engine: is a collection of various pluggable reasoning modules to handle the facts present in the repository as well as to produce composite contexts. Ontology Reasoning Module can use various kinds of logics to support inference; description logic, first order logic, temporal logic and spatial logic to name a few. Context Reasoning Module can use Fuzzy logic, Bayesian networks and neural networks to produce composite context, providing different power and expressivity. Sometimes a combination of both reasoning mechanisms is needed.

Table 1. Example Feature Tuples. Same Sensor ID is assigned to individual sensors in the same physical space. Sensor ID = 3 indicates Bedroom.

Sensor ID	Sensor Type	Feature ID	Value		Time Stamp
			Numeric Value	Quantized Value (Symbolic, Probability)	
3	1(Audio)	1(Intensity)	X (dB)	1 (Silent, 0.9) 2 (Moderate, 0.1)	Xxxxx
3	2(Video)	3(Motion Pattern)	NA	1 (Stable, 0.8) 2 (Regular, 0.2)	Xxxxx
3	2(Video)	6(Posture)	NA	2 (Lying, 0.9)	Xxxxx
3	2(Video)	7(Luminous Intensity)	Y (cd)	1 (TotalDark, 0.2) 2 (Dark, 0.8)	Xxxxx

4 CAMUS Context Model

While context entities are conceptual entities, the information provided by them is called the contextual information. This contextual information has its own syntactic and semantic meanings. Some of the context entities are the producers of contextual information while others are consumers or both. Contextual information gathered from atleast one sensor is called the ‘elementary context’ while ‘composite context’ is any combination of elementary contexts or elementary and composite contexts as shown in figure 2.

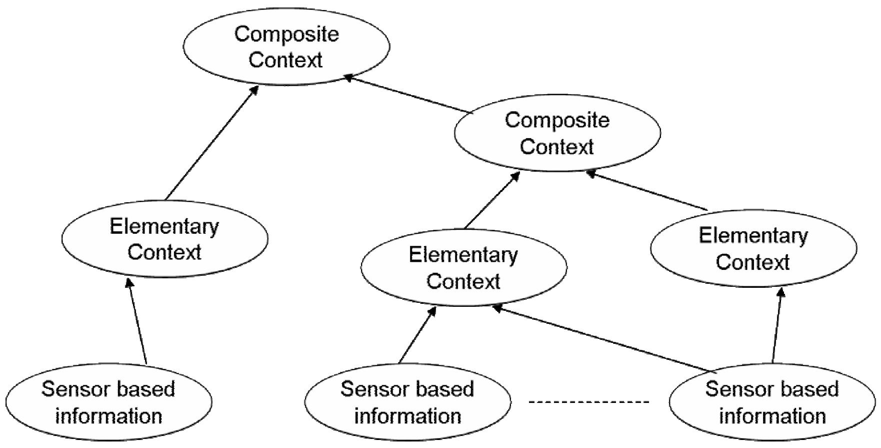

Fig. 2. Contextual information hierarchy

4.1 Basic Model

Diverse context entities ranging from various kinds of devices e.g. PDAs, mobile phones, ambient displays etc., running various applications, to various environment conditions e.g. sound intensity, light, temperature, traffic etc., are utilized by various kinds of agents e.g. software agents, persons, groups etc.

This variety leads us to categorize context entities, in our framework, mainly into agents, devices, environment, location and time. Location and time are kept separate from the other concepts to emphasize on the spatial and temporal aspects of the ubiquitous computing environment. These conceptual entities and their relationships are described in the ontology repository. Figure 3 shows the main context categories and few domain concepts of our context model, termed as, cont-el.

The shadowed ovals show, in figure 3 on next page, the main context categories while rectangles represent few of the concepts under the corresponding context category. Many new entities (devices, softwares etc.) may enter/leave the variable ubiquitous environment, but they can be made part of the system by adding their definitions at runtime into the ontology database and related to existing entities by various ontology language (OWL) constructs like subClassOf, disjointWith etc. So, representing context entities in the ontology brings all benefits of ontology world.

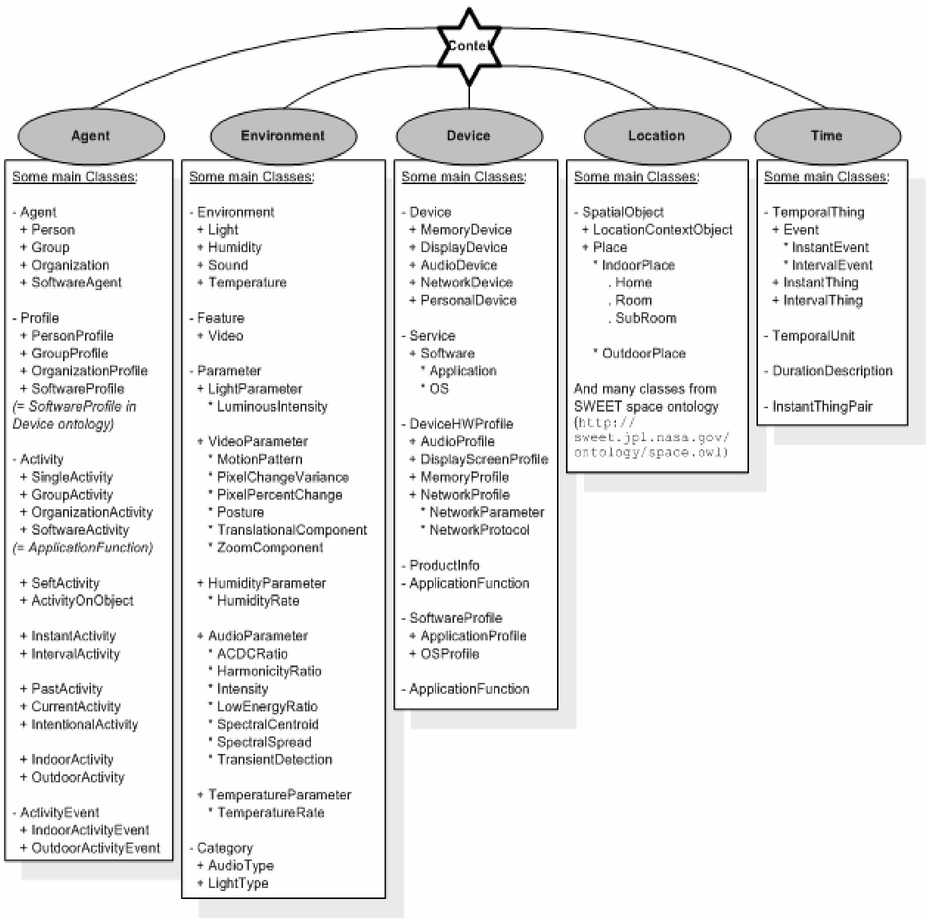

Fig. 3. Expandable Cont-el Basic Categorization and Some Domain Concepts

4.2 Detailed Model

Context entities and contextual information are described in the ontologies; facilitating various parts of the ubiquitous computing environment to interact with each other effectively. We have described ontologies for a home domain. The different ontologies made are based on basic categorization described above. In the following paragraphs, we will describe part of different ontologies for the home domain.

For the entities related to Agent, we have top level concept called Agent. It has been further classified into SoftwareAgent, Person, Organization, and Group. Each Agent has property hasProfile associated with it whose range is AgentProfile. Also, an Agent isActorOf some Activity. Activity class, representing any Activity, can be classified based on the Actor of it e.g. SingleActivity (which has only one actor), GroupActivity (which has Group as its actor and can have many SinlgeActivity instances). An Activity having some object of action on which it is done called ActivityOnObject like CookingDinner, TurnOnLight, or WatchingTV etc., while SelftActivity has no object of action e.g. Sleeping, or Bathing. Activity itself is not related to time and location but whenever activity happens, it generates an ActivityEvent (subclass of Event and LocationContextObject), encapsulating both time and location information.

The Device ontology is based on FIPA device ontology specification [11]. Every Device has properties of hasHWProfile, hasOwner, hasService, and hasProductInfo. Device is further classified into AudioDevice, MemoryDevice, DisplayDevice, NetworkDevice. PDA is considered here as subClassOf AudioDevice, DisplayDevice, NetworkDevice, MemoryDevice and PersonalDevice. All different devices have associated device profiles e.g. DisplayDevice hasDisplayProfile of DisplayScreenProfile containing properties resolution, color, width, height and unit. The hasService property of Device class has Range of Service. Service, in our framework, has at present Software subclass which is further sub-classified into disjoint classes Application and OS.

The environmental context is provided by the various classes in the Environment ontology. Humidity, Sound, Light and Temperature are different environmental information we are utilizing in our framework. This sensed information is available though different sensors deployed in the smart environment, and used by the applications to adapt their behavior. An Environment is unionOf all different variables (temperature, light, sound and humidity) mentioned above. Each of them has hasParameter property which links them to the different information gathered from environment. For Sound, the hasParameter has the range of AudioParameter class, which has subclasses, namely, ACDCParameter (ACDC stands for Average Crossing Direction Change), HarmonicityRatio, Intensity, TransientDetection etc. VideoParameter has been classified into MotionPattern, PixelChangeVariance, PixelPercentageChange, Posture, ZoomComponent etc.

Location ontology, an important aspect of ubiquitous computing environment, has SpatialObject as its top level class. This class is equivalent of SpatialObject defined at NASA Jet Propulsion Lab space ontology[1]. We have imported this ontology into our space ontology, as it describes useful information related to spatial objects. Place is a SpatialObject and has IndoorPlace and OutdoorPlace as it two subclasses. Each Place has hasEnvironment property which describes the environment conditions like temperature, humidity etc. A Place is a isPartOf some other Place. As we have defined ontology for the home domain, we have concepts like BedRoom, BathRoom, DinningRoom and LivingRoom etc. in our ontology. SubRoom isPart of Room, and represents an interesting place inside room such as OnBed, BesideDinningTable, InFrontOfTV, InSofa etc. LocationContextObject is anything which can have location context, having properties of locatedIn, locatedNearBy, locatedFarAwayFrom etc.

[1] http://sweet.jpl.nasa.gov/ontology/space.owl#

```
...
<owl:Class rdf:ID="Activity"/>
<owl:ObjectProperty rdf:ID="generatesEvent">
 <rdfs:domain rdf:resource="#Activity"/>
 <rdfs:range rdf:resource="#ActivityEvent"/>
 <rdf:type
   rdf:resource="&owl;InverseFunctionalProperty "/>
 <rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="ActivityEvent">
 <owl:equivalentClass>
  <owl:Class>
   <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#InstantActivityEvent"/>
    <owl:Class rdf:about="#IntervalActivityEvent"/>
   </owl:unionOf>
  </owl:Class>
 </owl:equivalentClass>
 <rdfs:subClassOf
 rdf:resource="&contellocation;LocationContextObject"/>
</owl:Class>
<owl:Class rdf:ID="IntervalActivityEvent">
 <rdfs:subClassOf rdf:resource="#ActivityEvent"/>
 <rdfs:subClassOf
   rdf:resource="&conteltime;IntervalEvent"/>
</owl:Class>
<owl:Class rdf:ID="InstantActivityEvent">
 <rdfs:subClassOf rdf:resource="#ActivityEvent"/>
 <rdfs:subClassOf
   rdf:resource="&conteltime;InstantEvent"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="containsActivity">
 <rdfs:domain rdf:resource="#Activity"/>
 <rdfs:range rdf:resource="#Activity"/>
 <rdf:type rdf:resource="&owl;TransitiveProperty"/>
</owl:ObjectProperty>
...
```

```
...
<owl:Class rdf:ID="Environment">
 <owl:equivalentClass>
  <owl:Class>
   <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Humidity"/>
    <owl:Class rdf:about="#Light"/>
    <owl:Class rdf:about="#Sound"/>
    <owl:Class rdf:about="#Temperature"/>
   </owl:unionOf>
  </owl:Class>
 </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="Light">
 <rdfs:subClassOfrdf:resource="#Environment"/>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#hasParameter"/>
   <owl:allValuesFrom rdf:resource="#LightParameter"/>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Parameter"/>
<owl:Class rdf:ID="LightParameter">
 <rdfs:subClassOf rdf:resource="#Parameter"/>
</owl:Class>
<owl:Class rdf:ID="LuminousIntensity">
 <rdfs:subClassOf rdf:resource="#LightParameter"/>
</owl:Class>
<owl:Class rdf:ID="Bright">
 <rdfs:subClassOf rdf:resource="#LuminousIntensity"/>
</owl:Class>
<owl:Class rdf:ID="TotalDark">
 <rdfs:subClassOf rdf:resource="#LuminousIntensity"/>
</owl:Class>
<owl:Class rdf:ID="Dark">
 <rdfs:subClassOf rdf:resource="#LuminousIntensity"/>
</owl:Class>
...
```

Fig. 4. Few definitions from Activity and Environment ontology in OWL

Temporal information is ubiquitous in real world situations and also considered as common need for ubiquitous computing applications. For time, we are using the concepts from DAML-Time ontology [12]. TemporalThing, a general concept, has subclass of InstantThing, IntervalThing and Event. InstantEvents (subclass of Event and InstantThing) can be thought of points which don't have any interior points e.g. entering a room, turning the TV on, and turning lights off. While IntervalEvents (subclass of Event and IntervalThing) denote events, that span some interval of time e.g. watching movie, playing games, or attending the meeting. Every TemporalThing has begins and ends properties pointing to the InstantThing and denotes its beginning and end. inside relation is between IntervalThing and InstantThing stating that some instant is inside the interval. before indicates that some TemporalThing (sleeping) has its end before the beginning of some TemporalThing (waking up). More details of our different ontologies can be found at our website[2].

[2] http://ucg.khu.ac.kr/ontology/0.1/

5 Reasoning Mechanisms

The contextual information provided by the environment leads to only elementary contexts. Some contexts are useful only when they are combination of some elementary and/or composite contexts, and also need consistency of contextual information. Our framework supports various pluggable reasoning modules and developer of the context Aggregator services can exploit any kind of reasoning mechanism based on application requirements. These reasoning modules are broadly classified into ontology and context reasoning mechanisms, but here we will only discuss ontology reasoning as context reasoning is out of scope of this paper.

5.1 Ontology Reasoning Mechanisms

High valued ontologies depend heavily on the availability of well-defined semantics and powerful reasoning modules. The expressive power and the efficiency of reasoning provided by OWL, (the semantics of OWL can be defined via a translation into an expressive Description Logics (DL) [13]), make it an ideal candidate for ontology constructs. The facts gathered from context entities make a factual world in OWL, consisting of individuals and their relationships asserted through binary relations.

Ontology reasoning helps us to find subsumption relationships (between subconcept-superconcept), instance relationships (an individual i is an instance of concept C), and consistency of context knowledge base, provided by Racer [14] Server. In the design phase of formalizing the context entities, OWL reasoning services (such as satisfiability and subsumption) can test whether concepts are non-contradictory and can derive implied relations between concepts.

Let us take an example to see how ontology reasoning can help deducing implied context. In location ontology, the property locatedIn is a TransitiveProperty, and isPartOf is subProperty of locatedIn. So when knowing that Bilbo is locatedIn Bed, and Bed is a part of BedRoom which is part of Home, the system can deduce that Bilbo is locatedIn BedRoom and Home.

Another example is how to map between the features we receive from Feature Extraction layer to simple contexts. Different sub classes of Parameter class have different hasValue restrictions on sensorTypeID, featureID and quantiziedLevel. Receiving a feature tuple with sensorTypeID = 1, featureID = 1 and quantiziedLevel = 1, we can create an instance of class Parameter with them, and then use OWL Reasoner to infer that the new instance is of type AudioParameter, Intensity and Silence.

5.2 Context Reasoning Mechanisms

However, many types of contextual information cannot be easily deduced using only ontology inference. In addition to ontology reasoning, we can also use logic inference. A set of rules can be defined to assert additional constraints for context entity instances when certain conditions (represented by a concept term) are met.

Over the concepts and relations defined in Cont-el, we can do a lot of reasoning based on many types of logics, such as description logic, description temporal logic, and spatial logic. We will take a closer look at how Cont-el supports these kinds of reasoning.

The spatial reasoning is based on the Location ontology and Region Connection Calculus [15]. We can infer about the spatial relations among the symbolic representation of space, such as `spatiallySubsumes` or `isDisconnectedFrom` relation between two `SpatialObject`. Here we illustrate one of those RCC rules:

```
[ (?x spc:spatiallySubsumedBy ?z),
  (?z rcc:isDisconnectedFrom ?y).
    → (?x rcc:isDisconnectedFrom ?y) ]
```

Based on the Time concepts in our ontology, we can define a set of rules for temporal reasoning. Temporal relations e.g. `meets`, `before` etc. and their inverses e.g. `metBy`, `after` etc. are taken from [16]. So we can define a set of temporal reasoning rules like this example:

```
[instant-before:
     (?x rdf:type tme:InstantThing), (?x tme:at ?timeX),
     (?y rdf:type tme:InstantThing), (?y tme:at ?timeY),
     lessThan(?timeX,?timeY)
         → (?x tme:before ?y)]

[interval-before:
 (?x rdf:type tme:IntervalThing), (?x tme:ends ?xE),
 (?y rdf:type tme:IntervalThing), (?y tme:begins ?yB),
 (?xE tme:before ?yB)
         → (?x tme:before ?y)]
```

The inferred temporal and spatial contextual information can be used for higher level reasoning. For example, categorizing activities into `PastActivity`, `CurrentActivity` and `IntentionalActivity` help defining some more complex inference. Following are some rules taken from our current implementation: (note that each time before calling the time reasoner, we have to update the `at` property of `Now` – a special instance indicating the current time, an individual of class `NowInstantThing` – with the current timestamp).

To infer that an activity is PastActivity

```
[past-act:
 (?a rdf:type act:InstantActivity),
 (?a act:containsActivityEvent ?e),
 (?a rdf:type act:InstantActivityEvent),
 (?e tme:before ?n) ,
 (?n rdf:type tme:NowInstantThing)
         → (?a rdf:type act:PastActivity)]
```

If agent has Waken Up and is Bathing then the Oven will Reheat the Breakfast

In DLR_{US} syntax [17], this rule can be expressed like this:

$$((\mathtt{OvenReheatingBreakfast} \sqsubseteq \bot)\ \mathcal{U}\ (\mathtt{WalkingUp} \sqsubseteq \mathtt{PastActivity} \sqcap \mathtt{Bathing} \sqsubseteq \mathtt{CurrentActivity}))$$

And here is the realization in Jena rule syntax:

```
[reheat:
 (?a1 rdf:type tme:WakingUp), (?a1 rdf:type act:PastActivity),
 (?a2 rdf:type tme:Bathing), (?a2 rdf:type act:CurrentActivity)
         → [(?o act:isActorOf ?a3),
              (?a3 rdf:type acthome:ReheatBreakfast)
                  ← (?o rdf:type devhome:Oven),
makeInstance(?a, act:isActorOf, acthome:ReheatBreakfast, ?a3)]]
```

Such temporal concepts and relations can play a useful role in the reasoning about contexts. All concepts and relations are written using the Protégé 2000 [18] which allows writing vocabularies in OWL. At present, we are using the Jena Semantic web toolkit [19] to insert the context information as it allows parsing, managing, querying and reasoning the ontologies programmatically.

However, Jena has limited support for other types of inferences, for example default reasoning and uncertainty reasoning. So we are considering using some other reasoning mechanisms, such as Bayesian network for uncertainty reasoning, and the Theorist framework for default and abductive reasoning. Cont-el ontologies and current reasoning mechanisms over it can provide contextual information, as input, for those higher inferences.

6. Conclusion and Future work

One of the fundamental characteristics of context-aware systems is formalization of context models, expressing entities independent of any specific application. Although complete formalization is impossible but it can be applied to the degree allowed by the domain for relatively stable or invariant entities. Using ontologies to describe the entities formally support also knowledge sharing, reuse, and logical reasoning. We believe that formalizing domains should be seen as emergent phenomenon constructed incrementally, leading to the sharing of contextual information among heterogeneous context-aware systems. In this paper, we discussed how formal modeling is useful for heterogeneous ubiquitous computing environment and presented our ontology for the home domain. We also discussed few (out of many) of the reasoning capabilities provided once context models are formalized.

Our final goal is to formalize ontology for the home domain, which is a part of our "Smart Home" test bed. Another direction is exploiting fully the reasoning provided by spatial and temporal aspects of the ubiquitous computing environment.

References

1. M. Weiser: Scientific America. The Computer for the 21st Century. (Sept. 1991) 94-104; reprinted in IEEE Pervasive Computing. (Jan.-Mar. 2002) 19-25
2. M. Satyanarayanan: IEEE Personal Communications. Pervasive Computing: Vision and Challenges. (Aug. 2001) 10-17
3. Dey, A.K., et al.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Anchor article of a special issue on Context-Aware Computing, Human-Computer Interaction (HCI) Journal, Vol. 16. (2001)
4. S. S. Yau, F. Karim, Y. Wang, B. Wang, S.Gupta: Reconfigurable Context-Sensitive Middleware for Pervasive Computing. (Jul.-Sep. 2002) 33-40
5. Chen, G., Kotz, D.: A survey of context-aware mobile computing research. Technical Report TR2000-381, Dartmouth College, Computer Science, Hanover, NH. (Nov. 2000)
6. Winograd T.: Architectures for Context. In: Human-Computer Interaction (HCI) Journal, '01, Vol. 16.
7. J. Davies, D. Fensel, F. V. Harmelen: Towards the Semantic Web, Ontology-Driven Knowledge Management, John Wiley & Sons. (Nov. 2002)
8. W3C Web Ontology Working Group: The Web Ontology language: OWL. http://www.w3.org/2001/sw/WebOnt/
9. Klyne, G., Caroll, J. J.: Resource Description Framework Abstract Concept and Syntax. W3C Recommendation. (10 Feb. 2004)
10. Hung, N.Q., Shehzad, A., Kiani, S. L., Riaz, M., Lee, S.: A Unified Middleware Framework for Context Aware Ubiquitous Computing. In: EUC2004, Japan. (Aug. 2004)
11. FIPA Device Ontology Specification. http://www.fipa.org/specs/fipa00091/SI00091E.pdf
12. Hobbs, J. R.: A Daml ontology of time. http://www.cs.rochester.edu/~ferguson/daml/daml-time-nov2002.txt. (2002)
13. Baadar, F., Horrocks, I., Sattler, U.: Description Logics. Handbook on Ontologies. (2004) 3-28
14. Haarslev, V., Moller, R.: Racer: A Core Inference Engine for the Semantic Web. In: EON2003, Sanibel Island, Florida. (Oct. 2003)
15. D. A. Randell, Z. Cui, and A. Cohn. A spatial logic based on regions and connection. In: Principles of Knowledge Representation and Reasoning (KR92), San Mateo, California, Aug. (1992)
16. Allen, J. F., Ferguson, G.: Actions and Events in Interval Temporal Logic. Technical Report 521, The University of Rochester, Computer Science Department, Rochester, New York. (Jul. 1994)
17. Artale, A., et al.: The DLR_{US} Temporal Description Logic. In: International Description Logics Workshop (DL-2001), Stanford, CA, USA. (Aug. 2001)
18. Protégé Project. http://protege.stanford.edu
19. Jena: A Semantic Web Framework for Java. http://jena.sourceforge.net/