Developing Context-Aware Ubiquitous Computing Systems with a Unified Middleware Framework

Hung Q. Ngo, Anjum Shehzad, Kim Anh Pham,

Maria Riaz, Saad Liaquat, and S. Y. Lee

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

Abstract. Context-awareness is one of the fundamental requirements for achieving user-oriented ubiquity. In this paper, we present the design and approach to a middleware solution that expedites context-awareness in a ubiquitous computing environment. Context-Aware Middleware for Ubiquitous computing Systems (CAMUS) envisions a comprehensive middleware solution that not only focuses on providing context composition at the software level but also facilitates dynamic features retrieval at the hardware level by masking the inherent heterogeneity of environment sensors. Complexity is handled by providing 'separation of concerns' between environment features extraction, contextual data composition and context interpretation. Different reasoning mechanisms are incorporated in CAMUS as pluggable services. Ontology based formal context modeling using OWL is described. With a systematic approach, CAMUS is proved to be a flexible and reusable novel middleware framework.

1 Introduction

The vision of ubiquitous computing, with devices seamlessly integrated into the life of everyday users, and services readily available to users anywhere all the time [1], [2], is becoming a reality. A ubiquitous computing environment is characterized by a diverse range of hardware (sensors, user devices, computing infrastructure etc) and equally diverse set of applications which anticipate the need of users and act on their behalf in a proactive manner. One of the major goals of context-aware computing is to provide services that are appropriate for a person at a particular place, time, situation, etc. Certainly, sensing becomes an enabling technology but at the same time it induces heterogeneity.

Context-aware computing involves acquiring context from various sources such as physical/logical sensors, devices and repository; performing context deducing and interpretation; monitoring for context changes and detecting situation occurrence; then carrying out dissemination of context to interested consumers, triggering of events, or adaptation of services in a distributed and timely manner.

Different approaches have been proposed for building context-aware applications and services. Anind Dey et al [3] have built a Context Toolkit to support rapid prototyping of certain types of context-aware applications by providing a number of

reusable components. Many projects have adopted this Toolkit approach [4], [5] while others are developing a middleware infrastructure [6], [7], [8]. The latter is gradually showing us its vital advantages. An appropriate infrastructure for context-aware systems would provide uniform abstractions and reliable services for common operations, support for most of the tasks involved in dealing with contexts, and thus simplify the development of context-aware applications. It would also ensure that different computing entities in the environment have a common semantic understanding of contextual information for knowledge sharing [7], [8].

Beside the broad characteristics of middleware in common, a middleware for context-awareness should also:

1. Support for heterogeneous and distributed sensing agents. Make it easy to incrementally deploy new sensors and context-aware services in the environment.
2. Provide different kinds of context classification mechanisms, including rules written in different types of logic (first order logic, description logic, temporal/spatial logic, fuzzy logic, etc.) as well as machine-learning mechanisms (supervised and unsupervised classifiers). Different mechanisms have different power, expressiveness and decidability properties, and system developers can choose the appropriate logic that best meets the reasoning requirements of each context.
3. Follow a formal context model using ontology to enable syntactic and semantic interoperability, and knowledge sharing between different domains.
4. Provide facilities for applications to specify different behaviors in different contexts easily, as well as privacy policy and security mechanism.

Furthermore, in order to make the middleware expandable in different application domains, it is desirable to have a reusable framework in the middleware. However, current context-aware architectures do not provide a complete solution for all the essential requirements in context-aware computing. Most of them rely on proprietary a protocol, thereby set a barrier to interoperability of different systems, and exclude developers from reusing existing components in different domains. In addition, they lack of a formal context model, instead, application-specific models have been used making it difficult to share context data across heterogeneous systems. Also, most context-aware systems do not really make use of the different context sources, which is often the most important aspect in ubicomp environment. As a result, no mechanisms yet exist to support the deployment of new sensing agents in the environment, as well as to hide their heterogeneity.

In this paper we propose a unified middleware framework for context-aware ubiquitous computing, namely CAMUS, to address these shortcomings of the existing architectures. A unification interface for sensor access mechanisms and feature abstraction for sensor data provide an efficient separation of concerns between different sensing techniques and context formation process. Different reasoning mechanisms are incorporated in CAMUS as pluggable services. Ontology based formal context modeling using OWL [14] is described for a home domain. Feature Tuple Space is utilized to provide underlying services for managing extracted features

from sensors. With a systematic approach, CAMUS is proved to be a flexible and reusable novel middleware framework.

The remaining paper is organized as follows. First, we describe a smart home scenario as a use case of our middleware framework. Then we present the core architecture of CAMUS. Next, we describe the formal context modeling and reasoning mechanisms incorporated in CAMUS. Finally, initial implementation, discussion and future work are provided.

2 A Smart Home Scenario

To elaborate the functional aspects of the system described in the next section, consider the following scenario:

Hung wakes up in the morning at 7:00 am. His morning routine involves taking a bath, dress up, drink coffee, listen to morning news, have breakfast and leave for office on his car. A number of things in the above scenario can be automated in a timely manner if the context information of Hung and his environment is available e.g. when he wakes up, the bath-tub fills up with warm (or cool) water, coffee pot heats up coffee, the TV turns on and shows broadcast from the news channel, lights in rooms/bathroom turn on/off according to his location etc. Before leaving the house, he can be informed on this PDA or mobile phone if he needs to take a raincoat/umbrella or to take a different route to office owing to a traffic block on the normal route that might result in delay. Hung wears an RFID tag on his wrist.

The house is fitted with a number of sensors including several wireless microphones, long range RFID reader (or several short range readers deployed around the house), temperature, humidity and light sensors in the rooms. A camera fitted to the ceiling of living room/bedroom is used to detect the users' condition and state e.g. lying on the bed, sitting, bending up, etc. The sensors are connected to a machine[1] that runs our middleware infrastructure.

Electronic devices (lights, bathtub controls, coffee pot, TV etc) are connected to a home control system[2] either wirelessly (Bluetooth, IrDA) or standard Ethernet from where control commands are sent. An application executing on the master system controls the behavior of these devices. This application needs the person's context information in order to make his home a smart environment.

Upon availability of such context information, the application can issue control commands to fill the bathtub with hot (or cold) water and within due time, ask the coffee pot to make coffee. When it is detected that the person has finished bathing, the T.V. is turned on for him to listen to morning news on his preferred channel. His PDA meanwhile asks the middleware services for weather and traffic conditions (about the route that the person usually takes to office). This information is gathered from sources external to the system and fall under the focus of interoperability of middleware with existing useful services. When the person leaves the apartment, lights, TV etc are automatically turned off and as a final step, the PDA checks with

1 It is assumed that CAMUS is deployed on a machine (workstation, desktop etc) in the house.

2 This master system can be a desktop running some home control software and may or may not be the same as the one where our middleware is deployed.

the RFID reader if any milk cartons are left in the refrigerator or not (milk cartons have RFID tags) and notes down in the person's schedule if he should bring any milk on the way back home.

3 CAMUS Core Architecture

Acquiring the users input from the real world is one of the challenges in context-aware computing. However, the most interesting kinds of context are those that humans do not explicitly provide. With advances in sensing and automated means of perceiving the physical environment along with more efficient pattern recognition techniques, we can automatically collect much more implicit context. The environment in general contains a diverse nomenclature of sensors having different access mechanisms, different sensory data and dissimilar representation schemes of such data. This diversity leads to potential problems and complexity in design and implementation. Thus a mechanism is required, which serves to extract information from the heterogeneous sensors and present to the upper layers for deducing contexts, in a standardized and unified manner.

3.1 Feature Extraction from Sensors

In CAMUS we provide a unification interface for each sensing agent (named Feature Extraction, FX Agent), as depicted in figure 1. Unification interface is aware of the definitions of the feature constructs based on the sensor types. This aids in deciding which sensor values are to be extracted and aggregated from sensor outputs. At this lowest layer in the CAMUS design, containers that hold access mechanism implementations (or wrappers) are also provided for individual sensors, to hide the communication details and data polling frequency of sensors from the above layers.

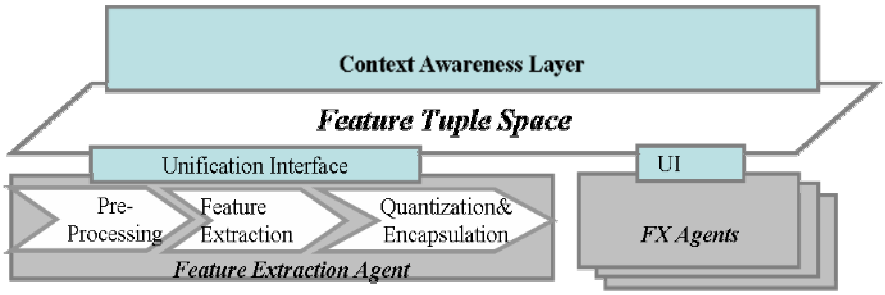

Fig.1. Feature Extraction Agent in CAMUS. Preprocessing module performs filtering, conversion, or contrast enhancement for sensor signals. Then features will be extracted, quantized/segmented, and encapsulated into feature tuples. Feature tuples are injected into Feature Tuple Space through Unification Interface for deducing context in upper layers

The extracted features should be as descriptive of the contexts they are attempting to model as possible. If the features are discriminative enough, the recognition

mechanisms would be simple and lightweight. In order to have a more expressive representation of context information, features are further quantized or segmented, resulting in a set of symbolic values that describe concepts from the real world. Fuzzy sets [9], [10] or crisp limits can be applied to quantize/segment the features. The probability (or confidence) associated with outputs of fuzzy quantization can be used as inputs of probability based context reasoning mechanisms [10], [11].

More informative sensors and a wider set of features would be essential for more detailed and accurate situational descriptions. Thus numerous useful features might be generated by each sensor; and the feature representation must facilitate the identification of individual features uniquely as well as collection of features by the same source. This is achieved by allocating a unique feature identifier in conjunction with the sensor and type identifiers to each Feature Tuple (FT) in the space.

```
FT= {Sesnor_ID, Type_ID, Feature_ID, Feature_Value,
Probability, Timestamp}
```

Consider the features in table 1, which are gathered from an audio and video sensor deployed in Bedroom of the Smart Home mentioned in our scenario. The information of audio sensor provided in the table can be encoded in Feature Tuples as follows:

FT_1= {3, 1, 1, 1, 0.9, xxxxx}

FT_2= {3, 1, 1, 2, 0.1, xxxxx}

Table 1. Example Feature Tuples. Same Sensor ID is assigned to individual sensors in the same physical space. Sensor ID = 3 indicates Bedroom. The semantic labeling and context reasoning processes are discussed in the next sections.

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

The main features extracted following the MPEG-7 specifications for audio and video signals in CAMUS are shown in table 2, along with their semantic meanings.

In CAMUS, Feature Tuple Space (FTS) is employed as underlying communication and storage mechanism. Feature Tuple Space provides a domain-wide persistent space where features gathered from the diverse sensors are stored with a common representation scheme. Features are stored directly as objects independent in space and time and decoupled from the generating processes; an important advantage in the ubiquitous environment in terms of interoperability and scalability. Various sub-

modules for feature extraction and context formation dynamically interact in the middleware by mere flow of objects in and out of the FTS.
The CAMUS provides different modules at the context layer to acquire various kinds of contextual information and reason about it in an appropriate way, as depicted in figure 2.

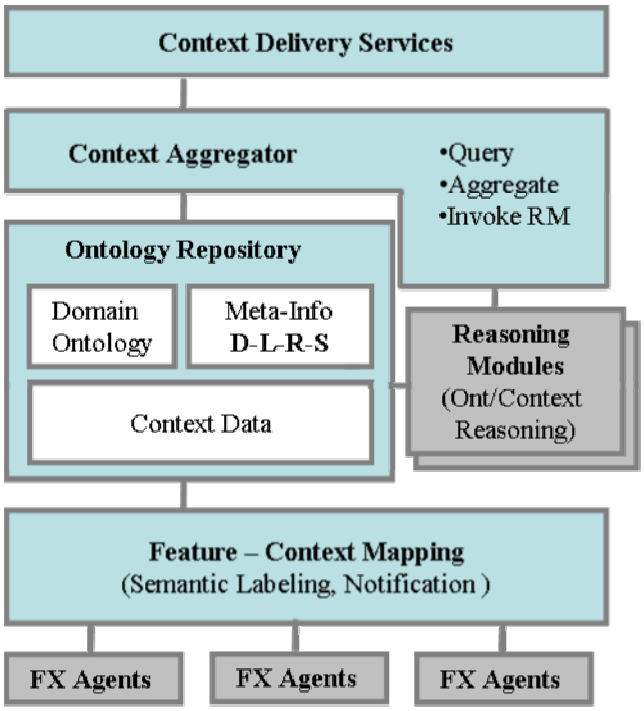

Fig.2. CAMUS Core Architecture.

3.2 Feature - Context Mapping

Upon the notification of feature change in the tuple space, this layer performs the mapping required to convert a given feature into context. The Feature - Context Mapping meta-information is saved in the ontology repository (section 4.2.2). Information such as user Id mapped to his name as well as his profile is saved in the ontology as a meta-information which enables this layer to do necessary mappings. If certain context is not present in the ontology repository and is requested by either context aggregator or reasoning module, it will register to the Feature Tuple Space for the feature, corresponding to that context. For example, the feature tuple in table 1

FT_1= {3, 1, 1, 1, 0.9, xxxxx}

would be mapped to corresponding context information

{Location.Bedroom, Environment.Sound.Intensity = Silent, Probability = 0.9, TimeStamp = xxxxx}

3.3 Ontology Repository

This main repository provides the basic storage services in a scalable and reliable fashion and contains the domain ontology, context information (including both elementary and composite context), and meta-information as explained below.

Domain Ontology. Domain ontology contains the domain concepts and properties with formal semantics described in OWL [14] and explained in detail in the context modeling.

Context Information. Here context is any information saved by the Feature - Context Mapping layer gathered from the environment through lower layers of the architecture and explained in detail in the context modeling.

Meta-Information. Having well structured meta-information about the various characteristics of the system allows flexibility and acts as a customizable solution for the specific needs of the use case. In our framework, we save meta-information about the devices (D in figure 2), sensors access mechanisms (S), feature to context labeling (L) (used by Feature - Context Mapping layer) as well as the meta-information about the input, output and capabilities of various pluggable reasoning modules (R).

3.4 Reasoning Engine

Reasoning engine is nothing but a collection of various pluggable reasoning modules. Different applications and domains have different reasoning requirements and the reasoning engine provides a collection of pluggable reasoning modules, providing easy integration, knowledge maintenance and re-use. Reasoning engine has to handle the facts present in the repository as well as to produce composite contexts.

Ontology Reasoning Module. This module is concerned with the class descriptions (domain ontology) and instance data (context information) of the ontology repository. This module is necessary to provide the entailed knowledge not formally present in the repository and also uses axioms and rules associated with the reasoner. This reasoner uses various kinds of logics to support inference; description logic, first order logic, temporal logic and spatial logic to name a few.

Context Reasoning Modules. The reasoning engine can contain one or more context reasoning modules based on application requirements. The need for context reasoning modules arises because not all information can be gathered from sensors. Sometimes, information from multiple sources is required to produce composite context. Sometimes, a combination of context and ontology reasoning can be used. Once composite context is saved in the repository; it is just a normal context like other elementary contexts. Fuzzy logic, Bayesian networks and neural networks can be used to produce composite context, providing different power and expressivity.

3.5 Context Aggregator

Context aggregation service is responsible for satisfying certain context queries. Each context aggregation service performs a specific function. An example service can be detecting that user has awaken and performing certain actions. Based on required

context information, it can either utilize the ontology reasoning module or context reasoning module or both. Upon detecting that certain context is composite one, it will retrieve meta-information from the repository about the specific context reasoning module providing the composite context. Once retrieved, it will invoke the corresponding reasoning module and return the result back to the application requesting the context.

3.6 Context Delivery Services

The context delivery services perform the job of searching appropriate context aggregators and delivering them to the applications. These include registration, query and notification services [19], [20]. Context Aggregators register with the registration service to provide the information about the context they can deliver. Interested applications and agents query the registration service to find services of their interests. The registration service upon finding appropriate aggregator, returns the handler to the requested clients.

Each context aggregator specifies the context it provides, by utilizing the concepts defined in the ontology repository. This standard schema sharing allows the different kinds of entities to be described and utilize by registration service to find useful services needed by the applications, thus allowing a flexible mechanism for exchanging descriptive information of various entities. In our framework, this semantic matchmaking [21], [22] is based on querying the Racer [23] Server which allows subsumption and classification of different concepts defined in the ontology.

4 Formal Context Modeling

Once features are gathered from the feature extraction layer, the next job is to have a global picture of the environment variables (features) at the context layer. Context [12], in general words, is any information which is used to help applications more adaptive to the surrounding environment and more responsive to the user. Here, we should mention that context is not any information gathered from the environment through sensing technology but the logical context of applications as well.

With the importance of context comes the issue of how to represent the context effectively. There are many context modeling techniques being used in context aware computing. Name value pairs [3] and entity relation model, to represent context, are easier to implement but lack consensus on semantics of the representation. Modeling context as objects [13] with fields containing state of context and methods/functions to access, modify or register for changes to context also suffer from the same problem. So, to represent and manage context information in a systematic manner, we need a common shared understanding of the context. For this purpose, we are using the W3's Web Ontology Language (OWL) [14].

4.1 Using OWL as formal Context Modeling

To enable sharing and reuse of context, we need to describe the domain at hand in a formal way. For this purpose, ontologies are used, defined as a shared and common understanding of a domain that can be communicated between people and application system [15]. Informally, they are used to represent the vocabularies of a domain describing important concepts and relationships among them. Since, ubiquitous computing environment is characterized by various domains e.g. home, office, university etc; ontology can play a useful role in sharing the domain knowledge of a particular environment.

OWL enables us achieve this goal in two steps. First, it allows us to define concepts and relationships among concepts in a domain of discourse e.g. describing person, devices, location etc. Second, it allows us to define instance data pertaining to some specific time and space e.g. person A is carrying a device B while on the road. 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.

Since OWL is a knowledge representation language and has explicit semantics associated with the knowledge, among other advantages it brings to us is the reasoning capability which intelligent systems and agents can use to infer useful contexts. Also, it can be used to represent meta-information about the sensors and its profile, e.g. in our architecture, we are also using OWL to represent access mechanism to the sensors, and the associated policies.

4.2 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 at least 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 3.

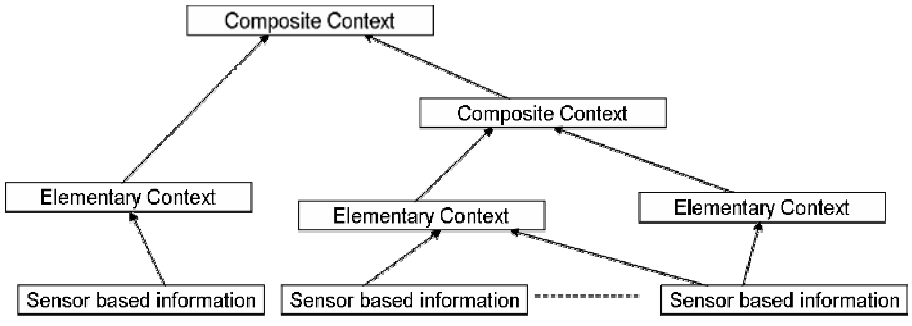

Fig.3. Contextual information hierarchy

4.2.1 Basic Model

Diverse context entities are usually found in the ubiquitous computing environment. These entities range 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., utilized by various kinds of agents e.g. software agents, persons, groups etc. All these entities are present and functioning at different times at different places.

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 4 shows the main context categories and few domain concepts of our context model, termed as, *cont-el*.

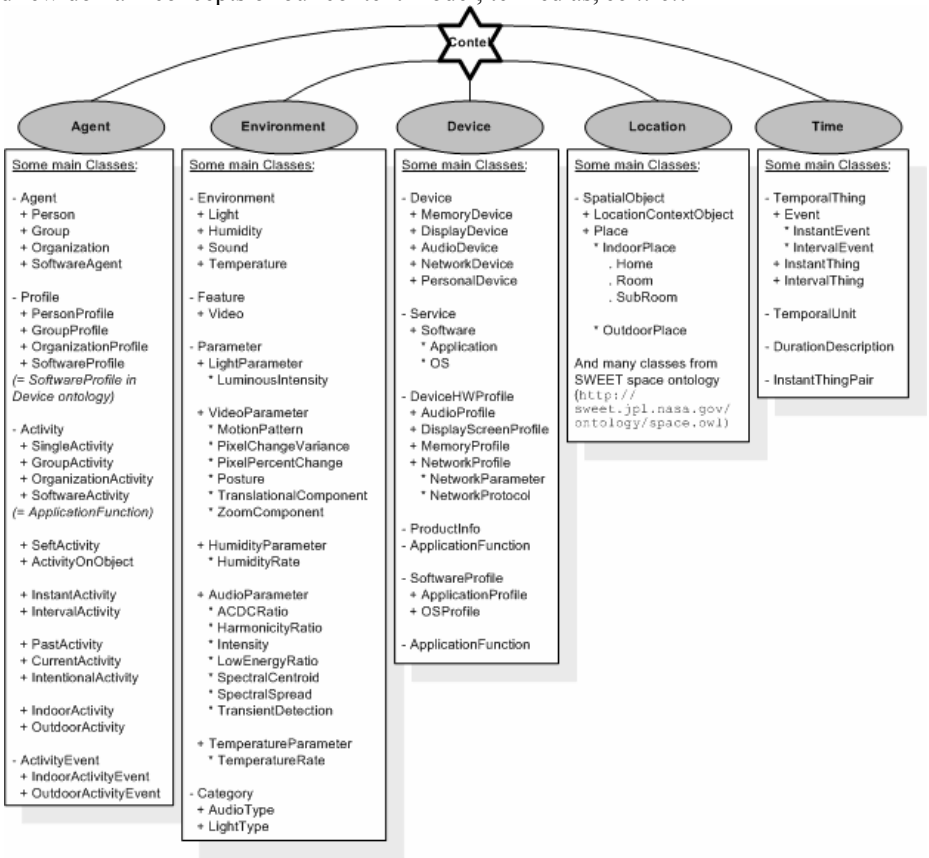

Fig.4. Expandable Cont-el Basic Categorization and Some Domain Concepts

The ovals represent the main context categories while shadow rectangles represent few of the concepts under the corresponding context category. Although dynamic environments result in addition of new entities, but they can added in the ontology and related to existing entities by various ontology language (OWL) constructs like

subClassOf, disjointWith etc. Also, if some entities change their meaning over the passage of time, they can be controlled through ontology versioning. So, representing context entities in the ontology brings all benefits of ontology mechanism.

4.2.2 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 different ontologies.

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.

Our Device ontology is based on FIPA device ontology specification [17]. 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 Ratio), `HarmonicityRatio`, `Intensity`, `TransientDetection` etc. `VideoParameter` has been classified into `MotionPattern`,

PixelChangeVariance, PixelPercentageChange, Posture, ZoomComponent etc.

```
...
<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.5. Few definitions from Activity and Environment ontology in OWL

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[3]. 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

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

etc. `LocationContextObject` is anything which can have location context, having properties of `locatedIn`, `locatedNearBy`, `locatedFarAwayFrom` etc.

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 [16]. `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[4].

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.

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) [24]), 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 [23] Server. In the design phase of formalizing the context entities, OWL reasoning

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

services (such as satisfiability and subsumption) can test whether concepts are non-contradictory and can derive implied relations between concepts.

Also, 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. For instance, in Home, a person is generally considered as not doing official work. This relationship is asserted by a rule (operator→) that fires for every individual that is classified as a member of concept Person and located in Home.

$$(\exists \text{locatedIn } \{\text{Home}\}.\text{Person}_x) \rightarrow (\neg \text{OnJob}.\text{Person}_x)$$

All concepts and relations are written using the Protégé 2000 [25] which allows writing vocabularies in OWL. At present, we are using the Jena Semantic web toolkit [26] to insert the context information as it allows parsing, managing, querying and reasoning the ontologies programmatically.

5.2 Context Reasoning Mechanisms

Besides rules written in some form of logic, CAMUS also uses various machine learning techniques to deal with context. Learning techniques that can be used include Bayesian learning, neural networks, reinforcement learning, etc. Reinforcement learning or neural networks could be used to learn the appropriate action to perform in different states in an online, interactive manner. For learning the conditional probabilities of different events, Bayesian learning is appropriate. Currently we are focusing particularly on Bayesian networks, because they provide a flexible, noise resilient and intuitively interpretable framework for classification, as well as causal modeling [10], [11] and have proven useful in many application domains.

The naïve Bayes classifier is used to infer user activities at home. It uses context data described by the ontology, a vector of elementary contexts with their confidence values as inputs. No background information modeling is required, except for choosing the relevant network inputs. Fuzzy membership values can be applied as virtual evidence [11]. The classifier has proven robust even with missing, uncertain, and incomplete information, and especially computationally efficient. Based on context data, we seek to learn probabilistic classifiers for relevant user activities, i.e.:

$$a \in ACT = \{\text{Sleeping, Walking, WatchingTV, } \ldots\}$$

Using a Bayesian approach, the system can determine the activity a_{MAP} with the maximum a posteriori (MAP) probability:

$$a_{MAP} = \arg\max_{a \in ACT} P(a \mid \text{Location, Motion, Posture, Environment\_params, Feature\_params})$$

We can easily look for the maximum element by applying the Bayesian theorem (the constant denominator can be eliminated because of the argmax), or giving some other simplify assumptions such as all sensor value are conditionally independent (naïve Bayes classifier). Figure 6 illustrates this recognition process with the input values described in table 1.

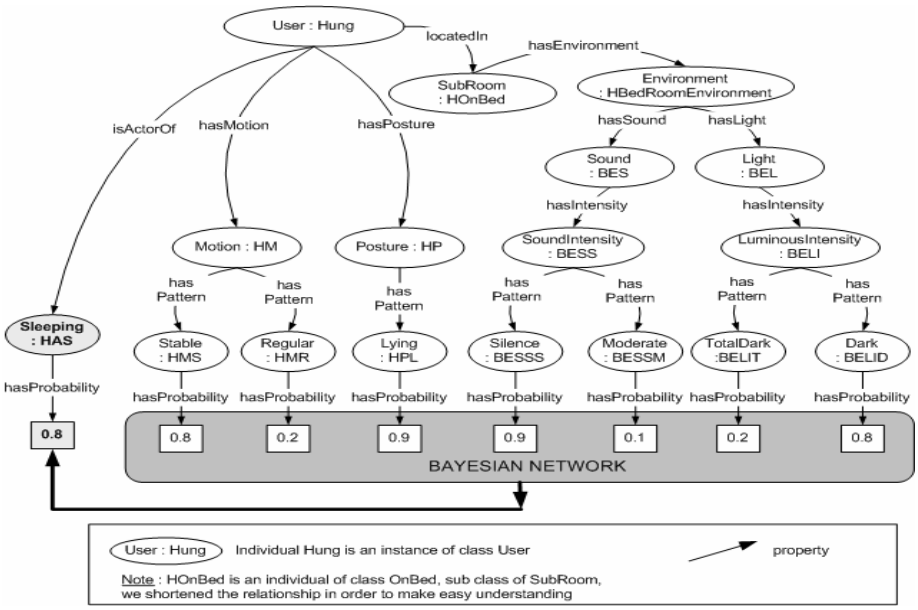

Fig.6. An example of deducing user activity using Bayesian network

6 Initial Implementation

In the Feature Tuple Space, we are using the Tspaces [28] from IBM. Tspaces provides considerable benefits because of its small footprint, support for group communication, access control and flexibility for adding customized operations and data types in the tuple space. Additional database features such as data integrity, indexing, transactions and event notifications are also available for improved search performance and reliability.

We are using the MySQL database as the ontology repository. All concepts and relations are written using the Protégé 2000 [25] which allows writing vocabularies in OWL. At present, we are using the Jena Semantic web toolkit [26] to insert the context information as it allows parsing, managing, querying and reasoning the ontologies programmatically. The Feature - Context Mapping layer utilizes the API provided by Jena to insert the elementary context into MySQL. An in-memory or flat file model can be used instead of database if the context domain is small and contains not many contexts to manage. The rule based reasoning engine provided by Jena uses RDQL (RDF Query Language), to access the stored knowledge in the ontology repository. Naïve Bayes approach is used to detect that the user activity using the continuous elementary context from the ontology repository and the produced composite context is saved in the ontology repository once inputs to the network change.

In order to provide customized wrappers for new sensors, a developer will extend from the basic functionality provided by the infrastructure along with providing a

description of the sensor in OWL. This description (meta-info) is used by the Feature - Context Mapping layer to perform necessary mappings for production of context.

Table 2. Expandable Features implemented in CAMUS, both simple and composed features are extracted to combine with other sensors' features for deducing context data.

Sensor Type	Feature Type	Feature Value (Quantized**.**Symbolic value)
1.Audio	1.Intensity	{1.Silent, 2.Moderate, 3.Loud}
	2.ACDCRatio (Average Crossing/Direction Change Ratio)	{1.VeryLow, 2.Low, 3.Avg, 4.High, 5.VeryHigh}
	3.HarmonicityRatio	{1.Low, 2.Medium, 3.High}
	4.SpectralCentroid	{1.Ultralow, 2.Low, 3.Medium, 4.High}
	5.SpectralSpread	{1.Low, 2.Medium, 3.High}
	6.TransientDetection	{0.None, 1.Transient}
	7.LowEnergyRatio	{1.Low, 2.High}
	8.AudioType	{0.Unknown, 1.Music, 2.TelephoneRing, 3.Applause}
2. Video	1.Pixel Percent Change	(%) OR {1.Low, 2.High}
**Posture is a composed feature, determined from the basic video features such as pixel change variance, translational and zoom components of motion*	2.Pixel Change Variance	{1.Low, 2.Medium, 3.High}
	3.Motion Pattern	{1.Stable, 2.Regular, 3.Irregular}
	4.Translational Component	{0.None, 1.Up, 2.Down, 3.RightSideway, 4.LeftSideway}
	5.Zoom Component	{0.None, 1.ZoomIn [Forward], 2.ZoomOut [Backward]}
	6.Posture	{1.Standing, 2.Lying, 3.Bending, 4.Walking}
	7.Luminous Intensity	{1.TotalDark, 2.Dark, 3.Bright, 4.VeryBright} OR (% luminance)
3.Light	1.Light_Source	(0.NotDetermined, 1.Natural, 2.Artificial)
4. Temperature	1.Temperature	{1.Cold, 2.Warm, 3.Normal, 4.Hot}
5.Humidity	1.Humidity	{1.Dry, 2.Normal, 3.Humid}
6.RFID	1.Tag_ID	{PersonID, DeviceID etc}
	2.Reader_ID	{1.Washroom, 2.Kitchen, 3. Bedroom, 4.Living room…}
	3.Granularity	{1.Low, 2.Medium, 3.High}
{7.TV, 8.ElectricOven, 9.MainDoor, 10.WaterHeater}	1.Status	{1.On, 2.Off} (planned for future implementation)

7 Discussion and Future Work

Since features are quantized/segmented to semantic value, feature tuples can greatly reduce the communication overheads. Thus, communication facilities with small footprint like emerging ZigBee technology can be employed exchanging features among interested parties. This can promote the development of autonomous, distributed and low cost, plug and play sensing modules with embedded feature extraction and classification algorithms.

OWL is the effort towards standard ontologies. It requires an effort to agree upon concepts and relationships in different domains to enable sharing the context. The semantic interoperability should be seen as an emergent phenomenon constructed incrementally, leading to the web of emergent semantics. Same effort is required in the context-aware computing to agree upon common semantics among different domains so that context information can be shared in the real sense.

A 'Smart Home' test bed has been setup and a prototype CAMUS system is under development for step-by-step evaluation of the proposed architecture. This will enable us to verify the architecture, witness strong and weak spots, improve upon the architecture and provide effective solutions for real world applications.

Many prototype and commercial context-aware middleware are expected to coexist in a real deployment environment. CAMUS will provide interfaces for communication with middleware solutions by other developers. This cross platform communication enables data sharing and adds to the possible amount of context information available to CAMUS users. As a first step, the authors will provide OSGi [27] compliant interface/bundle for communication and data utilization from the Context Toolkit [3] infrastructure.

8 Summary

In this paper we have presented the CAMUS as a unified middleware framework for context-aware ubiquitous computing systems. CAMUS envisions a comprehensive middleware solution that not only focuses on providing context composition at the software level but also facilitates dynamic features retrieval at the hardware level by masking the inherent heterogeneity of environment sensors. Complexity is handled by providing 'separation of concerns' between environment features extraction, contextual data composition and context interpretation. Different reasoning mechanisms are incorporated in CAMUS as pluggable services, ranging from rules written in different types of logic to machine-learning mechanisms.. Ontology based formal context modeling using OWL is described for a home domain. With a systematic approach, CAMUS is proved to be a flexible and reusable novel middleware framework

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. Jang, W. Woo: Ubi-UCAM: A Unified Context-Aware Application Model. In: Context 2003, Stanford, CA, USA. (Jun. 2003)
5. J. Hong: The Context Fabric. http://guir.berkeley.edu/projects/confab/
6. Kumar, M.; Shirazi, B.A.; Das, S.K.; Sung, B.Y.; Levine, D.; Singhal, M: PICO: a middleware framework for pervasive computing. In: IEEE Pervasive Computing, Vol. 2 Issue 3. (July – Sept, 2003) 72- 79
7. Anand Ranganathan and Roy H. Campbell: A Middleware for Context-Aware Agents in Ubiquitous Computing Environments. In: CM/IFIP/USENIX International Middleware Conference, Brazil. (Jun. 2003)
8. Chen Harry, Tim Finin, and Anupam Joshi: An Intelligent Broker for Context-Aware Systems. In: Ubicomp 2003, Seattle, Washington. (Oct. 2003)
9. Zadeh, L.: Fuzzy Sets. Information and Control 8, (1965) 338-353
10. Korpipaa, P., Koskinen, M., Peltola, J., Makela, S. M., Seppanen, T.: Bayesian approach to sensor-based context awareness. In: Personal and Ubiquitous Computing, Vol. 7, Issue 2. (July 2003) 113-124
11. Pearl, J.: Probabilistic Reasoning in Intelligent Systems. Revised second printing. Morgan Kaufmann, San Francisco. (1988)
12. Winograd T.: Architectures for Context. In: Human-Computer Interaction (HCI) Journal, '01, Vol. 16.
13. S. S. Yau, F. Karim, Y. Wang, B. Wang, S.Gupta: Reconfigurable Context-Sensitive Middleware for Pervasive Computing. (Jul.-Sep. 2002) 33-40
14. W3C Web Ontology Working Group: The Web Ontology language: OWL. http://www.w3.org/2001/sw/WebOnt/
15. J. Davies, D. Fensel, F. V. Harmelen: Towards the Semantic Web, Ontology-Driven Knowledge Management, John Wiley & Sons. (Nov. 2002)
16. Hobbs, J. R.: A Daml ontology of time. http://www.cs.rochester.edu/~ferguson/daml/daml-time-nov2002.txt. (2002)
17. FIPA Device Ontology Specification. http://www.fipa.org/specs/fipa00091/SI00091E.pdf
18. Brickley, D., Miller, L.: FOAF Vocabulary Specification. http://xmlns.com/foaf/0.1/
19. Jini. http://www.jini.org/
20. UPnP. http://upnp.org/
21. Trastour, D., Bartolini, C., Gonzalez-Castillo, J.: A Semantic Web Approach to Service Description for Matchmaking of Services. HP Labs Bristol. HPL-2001-183. (2001)
22. Li, L., Horrocks, I.: A software framework for matchmaking based on semantic web technology. In: WWW 2003, ACM. (2003) 331-339
23. Haarslev, V., Moller, R.: Racer: A Core Inference Engine for the Semantic Web. In: EON2003, Sanibel Island, Florida. (Oct. 2003)
24. Baadar, F., Horrocks, I., Sattler, U.: Description Logics. Handbook on Ontologies. (2004) 3-28
25. Protégé Project. http://protege.stanford.edu
26. Jena: A Semantic Web Framework for Java. http://jena.sourceforge.net/
27. OSGi. http://www.osgi.org/
28. IBM Research: TSpaces. http://www.almaden.ibm.com/cs/TSpaces