특집

Middleware for Context-Aware Ubiquitous Computing

Hung Q. Ngo* Sungyoung Lee**

목 차

1. Introduction

1.1 Ubiquitous Computing Vision

The term "Ubiquitous Computing" was originally introduced by Mark Weiser[1] in the year 1991. In his fundamental article "The Computer for the 21st Century"[2],he elaborated about "the computer that disappears". For Weiser the way into the 21st century was obvious: Computer and Network technologies are getting to be smaller, cheaper, and more powerful. Therefore, more and more everyday artifacts are going to be equipped with a reasonable amount of computing power and, maybe even more important, are networked together into a virtually unique network of communicating "things that think". In the pure sense of the word, computing gets "ubiquitous", anywhere, any time. Computers in every thing that is calmly doing what we intend it to do, in a way that is non-obtrusive and user-friendly, in a sense that we do not have to focus our attention on the trivia of running an electronic system.

Research on Ubiquitous Computing(Ubicomp) is related to very many other disciplines from Robotics and Embedded Systems, Networking and Distributed Systems, to Artificial Intelligence and Technology&Society. Thus Ubiquitous computing is a very difficult integration of human factors, computer science, engineering, and social sciences.

1.2 Context-aware Computing

One goal of Context-aware Computing is to acquire and utilize information about the context of a device to provide services that are appropriate to the particular people, place, time, events, etc. For example, a cell phone will always vibrate and never beep in a concert, if the system can know the location of thecell phone and the concert

* a Master student in the Computer Engineering Department at Kyung Hee University
** Professor in the Department of Computer Engineering, Kyung Hee University

schedule. Often, the term "Context-Aware Computing" is used in a sense synonymously to Ubiquitous Computing. This is because almost every Ubicomp application makes use of some kind of Context. Ubicomp is mainly about building systems which are useful to users, which "...weave themselves into the fabric of everyday life until they are indistinguishable from it"[2]. Ubicomp systems Context is essential: "How can a system be helpful for a user?" Users tend to move around often, doing new things, visiting new places, changing their mind suddenly, and changing their mood, too. Therefore, a helpful system seems to need some notion of Context.

In the Human point of view, we have a quite intuitive understanding of Context. Here, Context is often referred to as "implicit situational understanding." In social interactions Context is of great importance: A gesture, a laugh, or the intonation of sentences is building up the implicit "picture" of what is meant or what my communication partner is thinking. The same words can have adifferent meaning in different contexts.

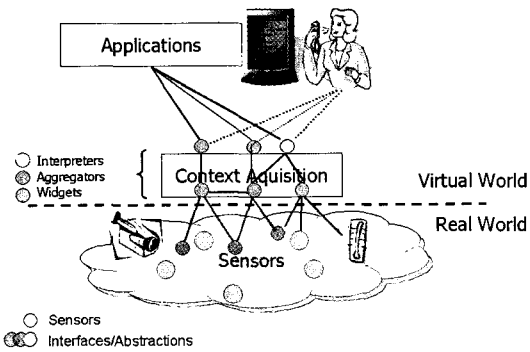
In Computer Science Context is quite familiar. Be it within the discipline of Artificial Intelligence ("Thinking machines"), in Robotics ("Adaptive Systems"), in User Interface Design (like adaptive UIs or office assistants like the Microsoft Office assistant called "Clippy"), or basically any other discipline (to some extent). Especially, every discipline dealing with human users tries to take into account human behavior one way or the other, with the generated output loops back as part of the vector of input values.

From the variety of definitions commonly used by Ubicomp researchers we can imagine how difficult it is to find a common ground. Context definitions are far away from mathematical precision and a particular definition often strongly depends on an authors' subjectiveness:

- ⊙ Schilit and Theimer [3]: "Context is location, identities of nearby people and objects, and changes to those objects."
- ⊙ A. Dey and Abowd [4]: "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 applications themselves."
- ○ Pascoe [5]: "Context is the subset of physical and conceptual states of interest to a particular entity."

So what is this leading to? Are those definitions helpful or misleading? In the sense of a functional definition they are only helpful as a general description of what to do. As an application designer they are only stating what I am doing anyway: Trying to figure out what input is needed to produce the desired output. Hence, it is of topmost importance to have some common ground or a common "vocabulary" when talking about what context is. We need some sort of formal approachtowards handling and describing Context. Furthermore, in a software engineering sense, we need building-blocks for building context-aware applications in a structured way. The Context Toolkit [6] by A. Dey is a step into this direction and a good example for this principle

(fig.1.) The Toolkit includes building blocks called "Widgets", wrapper classes for Sensors which serve as a hardware-abstraction layer, "Aggregators", which concentrate multiple input values to a single output value, and "Interpreters", implementing some application logic and generating application dependant "higher-level" output based on the input given. They interpret the incoming data according to a pre-programmed scheme.



(Figure 1) The Context Toolkit Core Components

With the Context Toolkit, the development of Context-aware applications basically consists of several distinguishable steps including 1) The real-world is sensed; 2) Context is detected, aggregated, "interpreted", and 3) Applications are custom-built to match the "context-detection" technology. However, we believe that there is more tool-support necessary for software engineering and the design of Context-aware applications than provided today. We want to emphasize that the way applications are developed is very dependant on the underlying technology used, which we consider as bad practice in the long run. Research in the direction of decoupling applications from data acquisition seems to be important. This is detailed in the section 2, Middleware for Context-aware Ubiquitous Computing Environments.

1.3 Middleware and Its Evolution

The role of middleware is to ease the task of designing, programming and managing distributed applications by providing a simple, consistent and integrated distributed programming environment. Essentially, middleware is a distributed software layer, or 'platform' which abstracts over the complexity and heterogeneity of the underlying distributed environment with its multitude of network technologies, machine architectures, operating systems and programming languages[7].

Different middleware platforms support different programming models. Perhaps the most popular model is Object-based Middleware in which applications are structured into (potentially distributed) objects that interact via location transparent method invocation. Prime examples of this type of middleware are the OMG's CORBA [7] and Microsoft's Distributed COM[7]. Both of these platforms offer an interface definition language(IDL) which is used to abstract over the fact that objects can be implemented in any suitable programming language, an object request brokerwhich is responsible for transparently directing method invocations to the appropriate target object, and a set of services (e.g. naming, time, transactions, replication etc.) which further enhance the distributed programming environment.

Sun's Jini[8] may also be categorized into

object oriented middleware. In Jini, service objects register with a centralized lookup-service which plays the role of matchmaker between clients and services. After a client finds a service, all interactions are performed in a location-transparent manner and without the aid of the lookup-service. Typically, object-based systems assume that a connection between a client and a service object is long-lasting, and therefore these systems do not address the possibility of disconnection in ubiquitous and mobile computing.

Not all middleware is object based, however. Two other popular paradigms are event based middleware and message oriented middleware both of which mainly employ 'single shot'communications rather than the request-reply style communication found in object based middleware. Event based middlewareis particularly suited to the construction of non-centralized distributed applications that must monitor and react to changes in their environment. Examples are process control, Internet news channels and stock tracking. Ubiquitous and Mobile Systems cover applications characterized by the heterogeneity of systems and devices, as well as the (spontaneous) patterns of interconnection. Due to the unpredictability of interaction schemes and the preferred use of asynchronous communication patterns, system design based on the notion of events seems to be superior to classical client/server interaction schemes. It is claimed that event based middleware has potentially better scaling properties for such applications than object based middleware. Message oriented middleware, on the other hand, is biased toward applications in which messages need to be persistently stored and queued. Message oriented middleware supports data exchange and request/reply style interaction by publishing messages and/or message queuing in a synchronous and asynchronous (connectionless) manner. Workflow and messaging applications are good examples.

The issues of mobile devices (heterogeneity, scare resources), network connection (limited bandwidth, high error rate, higher cost, and frequently unpredictable disconnections), and mobility (dynamic changes of the environment parameters) of ubiquitous and mobile systems posed new challenges to the design of middleware systems. The middleware need to be designed to achieve optimal resource utilization, adaptivity and dynamic reconfiguration. Reflective middlewareis concerned with applying techniques from the field of reflection in order to achieve flexibility and adaptability in middleware platforms. Reflectionis the capability of a systemto reason about and act upon itself. A reflective system contains a representation of its own behavior, amenable to examination and change, and which is causally connected to the behavior it describes. "Causally-connected" means that changes made to the system's self-representation are immediately reflected in its actual state and behavior, and vice-versa.

Tuple Space Middleware systems exploit the decoupled nature of tuple spaces for supporting disconnected operations in a natural manner. By default they offer an asynchronous interaction paradigm that appears to be more appropriate for

dealing with intermittent connection of mobile devices, as is often the case when a server is not in reach or a mobile client requires to voluntary disconnect to save battery and bandwidth. By using a tuple-space approach, we can decouple the client and server components in time and space. In other words, they do not need to be connected at the same time and in the same place. However, since JavaSpaces [9] and TSpaces [10], the common Tuple Space Middlewares, typically require at least 60Mbytes of RAM, they are not affordable by most handheld devices available on the market nowadays.

While traditional middleware for Distributed and Mobile environments do provide the basic mechanisms for different entities (or agents) to communicate with each other, they fall short in providing ways for agents to be context-aware. The ultimate goal of middleware for traditional computing environments is providing complete transparency of the underlying technology and the surrounding environments. Such an approach does not work for pervasive computing applications because being aware of the surrounding environment is the key to their effectiveness as we stated earlier. Moreover, Ubiquitous Computing environments feature a large number of autonomous agents. Various types of middleware (based on CORBA, Java RMI, SOAP, etc.) have been developed that enable communication interoperability between different entities, however, existing middleware have no facilities to ensure semantic interoperability between the different entities. Since agents are autonomous, it is infeasible to expect all of them to attach the same semantics to different concepts on their own. This is especially true for context information, since different agents could have a different understanding of the current context and can use different terms and concepts to describe context. A middleware for context -awareness will address this problem by ensuring that there is no semantic gap between different agents when they exchange contextual information.

2. Middleware for Context-aware Ubiquitous Computing Environments

2.1 Motivation of Context-aware Middleware

Different approaches have been suggested for promoting context-awareness among agents. The Toolkit approach proposed by Anind Dey et al [6] provides a framework for the development and execution of sensor-based context-aware applications with a number of reusable components. The toolkit supports rapid prototyping of certain types of context-aware applications. The other approach is developing an infrastructure or a middleware for context awareness. As we stated before, Context-aware Computing involves acquisition of contextual information, reasoning about context and modifying one's behavior based on the current context. A Context-aware Middleware would provide support for each of these tasks and greatly simplify the tasks of creating and maintaining context-aware systems. Middleware can help in continuous data acquisition, analysis,

and pattern detection to infer higher level context, and let developers focus mainly on developing the applications'functionality rather than diverting their effort to hardware-specific issues. Besides, a middleware would be independent of hardware, operating system and programming language and provide uniform abstractions and reliable services for common operations. It would, thus, simplify the development of context-aware applications. It would also make it easy to incrementally deploy new sensors and context-aware agents in the environment. It would define a common model of context, which all agents can use in dealing with context. It would thus ensure that different agents in the environment have a common semantic understanding of contextual information. Finally, a middleware would also allow us to compose complex systems based on the interactions between a numbers of distributed context-aware agents.

2.2 Functional Elements of Context Aware Computing

Context aware computing relies on multiple independent and cooperative enabling technologies. Based on the extraction of literature review, we have identified a set of necessary functional elements that a context aware system needs support essential context aware mechanisms. As we argued in previous section, only a general middleware approach can combine independent functional elements in context aware computing, and melt them into a coherent system to provide a complete solution. These functional elements include:

2.2.1 Context Sensing

In order to use context in services, there must be a mechanism to obtain the context data from diverse context sources. For example, the indoor location of a user can be obtained from an Infrared location sensor system, which detects the presence of a badge to conclude the location of the user wearing it. Context Sensing could be tightly-coupled with hardware sensors, while a component approach to decouple low-level sensing with high-level context usage can achieve reusable context sensing, thereby enabling the evolving of large scale context aware systems.

2.2.2 Context Model and Representation

Context model forms the foundation for expressive context representation and high-level context interpretation[11, 12]. Existing context models vary in the expressiveness they support and the types of context they represent, while their common considerations should be how to capture general features such as properties of an entity and interrelation between contextual objects. On the context representation layer, 'raw material' of context is transformed into a machine-readable format based on the context model. This is actually an abstraction layer that acquires sensor data from context sources, and then annotates raw data with semantics that are structured around a set of contextual entities (e.g., 'user', 'location' and 'device'.) and the relations (e.g., 'locatedIn') that hold between them. A uniformed context models is needed to facilitate context interpretation, context sharing and semantic interoperability.

2.2.3 Context Aggregation

Context information is obtained from an array

of diverse information sources. A centralized context aggregation mechanism can provide a persistent storage for distributed context, guarantees integrity of context, and offers shared mechanisms, relieving context-aware services from overheads caused by querying from distributed sensors[13]. When context is represented based on shared context model, context aggregation provides a foundation to merge interrelated information and enables further data interpretation.

2.2.4 Context Query

To explore general means of access to interrelated context spread across distributed context repositories, we need a high-level mechanism for context -aware services to issue queries without explicitly handling underlying data manipulation[14]. For example, a notification service for conference attendees require context like "find a list of researchers in this hall whose publications are in the same session with mine". The low-level operations of such complex context retrieval task should not be exposed to end users. Context query poses design issues such as context query language, event notification, and query optimization.

2.2.5 Context Reasoning/Inferring

Low-level information usually can not be directly understood and utilized by software services. Hence, there is a need to interpret low-level information and derive additional, high-level context. For example, a location based service wants to know the relative location with different levels of granularity (e.g., room-level, building-level, block-level) instead of sensor-driven position (the coordinates retrieved by the GPS system). In this case, we need to derive high-level location (e.g., 'which block is the user in?') from related contexts (e.g., GPS coordinates, the mapping between GPS coordinates and corresponding blocks). The context interpretation layer leverages reasoning/learning techniques to deduce high-level, implicit context needed by intelligent services from related low-level, explicit context.

For example, the rule-based reasoning engine can deduce user's current situation based on his location and environmental contexts. The inferred context might suggest that the user might be sleeping currently, since the time is 11 pm and he is staying at a dark, quiet 'Bedroom'. Another example of context interpretation could be machine learning based behavior prediction. The intelligent system could learn the pattern of user's actions from historical sequences of context data and then use this learned pattern to predict next event. For example, it could be predicted that once the user finished showering (turn off the electronic water heater) after 10:30 pm, he will check emails using the hand phone, and then go to bed after finishing reading them. Currently, context interpretation tasks are performed through various approaches including ad hoc interpretation, rule -based reasoning[15], and machine learning[16].

2.2.6 Context Discovery

In order for a context aware service to use a certain kind of context, there is a need for context requestors to find thesources providing it. The aim of context discovery is to locate and access

interested context sources in a self-configure manner. Issues of context discovery include service description, advertisement and event subscription[17].

2.2.7 Context-aware Application/Service

Finally, on the uppermost level (context utilization layer), context aware services utilize both low-level and high-level context to adjust their behaviors. The smart phone might then decide that the user probably does not want to answer any phone call when he is sleeping at home and forward those calls to the voice message box. It also could take account of predicted context and response by automatically adjusting the air-conditioning of the bedroom and downloading emails after the shower before sleep.

2.3 Some Middleware Architectures for Context-Aware Computing

A lot of work has been done in the area of context -aware computing in the past years, among which much of them are only concerned with one or more aspects in an ad hoc manner. In addition, most of them rely on proprietary a protocol, thereby set a barrier to interoperability of different systems, and exclude developers from reusing existing components. We are going to give a summarization of the most prominent projects in this research field.

2.3.1 Context Toolkit[6]

The seminal work of Context Toolkit developed a set of abstractions for sensors data processing in order to facilitate reuse and make context aware applications easier to build. Context Toolkit separates the low-level sensing from high-level applications, and introduces a middleware layer whose functionalities are collecting raw sensor information, translating it to an application-understandable format, and disseminating it to interested applications. In Context Toolkit, a "context widget"is a wrapper component that provides unified access interface to context. To handle context query and event notification, each context widget has a state that is a set of attributes and a behavior that is a set of callback functions triggered by context changes. The widget obtains raw contextual information from sensors and passes them either to interpreters or to servers for aggregation. Interpreters and servers use simple HTTP protocol for communication and the XML as the language model.

2.3.2 Solar [18]

The Solar system architecture proposed a graph-based abstraction for context aggregation and dissemination. The abstraction models the contextual information sources as event publishers, and context aware applications as event subscribers. A number ofevent processing and routing mechanisms are designed to avoid redundant computation at context aggregation and interpretation nodes, and reduce data transmission in large scale context aware systems. Applications use a subscription language to construct a logical event tree, based on event streams registered in a context -sensitive naming hierarchy.

2.3.3 Context Fabric[19]

Context Fabric is actually an extension of the pioneering work of ParcTab system[3]. It provides a distributed context-aware

infrastructure with two fundamental built-in services, namely event service and query service, to support the acquisition and retrieval of context data. Context Fabric uses an entity-relation styled logical context data model to represent the information about four kinds of concepts: entities, attributes, relationships, and aggregates. Context about each kind of entities are assigned network-addressable logical storage units called infospaces that can be directly queried from network. Context data in Context Fabric is encoded using XML, and stored in local file systems. XPath is utilized as the query language for addressing parts of the XML tree structure.

2.3.4 Gaia [15]

The Gaia project developed at the University of Illinois is a distributed middleware infrastructure thatprovides support for context aware agents in smart spaces. Gaia adopts a predicate model of context data to enable agents to be developed that use first order logic rules to decide their behavior in different contexts.

It also proposed that different logic reasoning and machine learning techniques can be adopted to support context interference according to different application requirements. DAML encoded ontologies is used to ensure semantic interoperability between different agents, as well as between different ubiquitous computing environments. All agents are implemented on top of CORBA, and use CORBA Naming Service and the CORBA Trading Service for service discovery.

3. Research Issues in Context-aware Middleware

Context aware computing has been drawing much attention from researchers for several years, the four projects described above is an example. However, context-aware services have never been widely available to everyday users. There are five reasons why context aware computing has been difficult to achieve. First, there is a lack of formal context model. Current context data models have generally been application-specific, making it difficult to share context data across heterogeneous systems. Simple and informal context models can not support expressive context representation and complex context interpretation [11, 12]. Second, context data come from diverse context sources such as hardware sensors and software services. For example, location information of a user can dynamically come from RFID, GPS, or cellular network system, while the location of a restaurant can be queried from a GIS system. This fact causes the heterogeneity issue.

Third, there is a lack of infrastructure support. Many of the former projects tightly coupled the sensing and reasoning tasks with the applications thus could not promote the flexibility and reusability of the software systems. Infrastructure approaches are needed to simplify the tasks of developing and maintaining context aware systems by providing high-level abstractions and reusable components for common operations [15, 19]. And finally, security issues and privacy concernsmust be addressed. The dynamism and ubiquity of the pervasive computing paradigm raise new issues for information security and user privacy.This is an especially difficult case because an important feature of context aware computing

is to share information across users and systems. Therefore, we need a well-established privacy mechanism that can balance context sharing and information security for context aware computing [11, 20]. We are going to discuss these issues in three main research sub-areas: Formal Context Modeling, Context Reasoning Mechanisms, and System Paradigm Design Factors.

3.1 Formal Modeling for Context Data

3.1.1 Why Formal Context Modeling is needed?

Context presentation is an important part of pervasive computing environments. Because context-aware applications must adapt to changing situations, they need a detailed model of users'activities and surroundings that lets them share users'perceptions of the real world. These entities may have different meanings associated with them in different pervasive environments. In order to have invariantmeanings 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, help to make a growing pool of well-tested context knowledge available to different context-aware systems.

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[6] and entity relation model, while others have used objects [21] 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.

3.1.2 Formal Context Modeling using OWL Ontologies

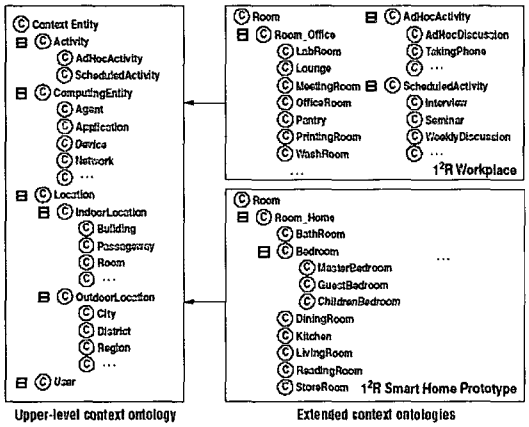
Context entities are the concepts in a domain of discourse, and to provide formal meaning of these concepts, ontologiesare used. Within the domain of knowledge representation, the term ontology refers to the formal, explicit description of concepts, which are often conceived as a set of entities, relations, instances, functions, and axioms, leading to shared and common understanding that can be communicated between people and application systems[22]. Formalizing domain not only contains the vocabularies of concepts but relationships among them as well. W3C's OWL (web ontology language)[23] 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 specifictime 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.

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 [24]), it can be used to represent meta-information about sensors, we can also use OWL to represent access mechanisms to the sensors and associated policies.

The following example shows the context ontology that describes a user named Hung.

```
〈User rdf:about="Hung"〉
〈name〉Hung〈/name〉
〈mbox〉nqhung@oslab.khu.ac.kr〈/mbox〉
〈homepage rdf:resource="http://ucg.khu.ac.kr
/~nqhung"/〉
〈office rdf:resource="#RoomB07"/〉
〈officePhone〉2493〈/officePhone〉
〈mobilePhone〉9999〈/mobilePhone〉
〈! More properties not shown in thisexample〉
〈/User〉
```



(Figure 2) An example of Context Ontology for Smart Spaces [25]

(Figure 2) presents an example of ontology system with upper-level (generic) context ontology and extended (domain specific) context ontologies for Smart Spaces[25].

3.1.3 Benefits of Semantic Web Ontology for Context Modeling

There are several potential advantages for developing context models based on Semantic Web Ontology. First, its expressiveness. Web ontology is modeled through an object-oriented approach, with expressive power entailed by their class/property constructors and axioms. Therefore it is more expressive than existing context models allowing us to capture more features of various types of context. Second, Knowledge Sharing. The use of context ontology enables computationalentities such as agents and services in pervasive computing environments to have a common set of concepts about context while interacting with one another. By allowing pervasive computing entities to share a common understanding of context structure, OWL ontologies enable applications to interpret contexts based on their semantics.

Third, based on ontology, context -aware computing can exploit various existing logic inferencemechanisms to deduce high-level, conceptual context from low-level, raw context,and to check and solve inconsistent context knowledge due to imperfect sensing. Because contexts described in ontologies have explicit semantic representations, Semantic Web tools such as federated query, reasoning, and knowledge bases can support context interpretation. Incorporating these tools into smart

spaces facilitates context management and interpretation. Fourth, Knowledge Reuse. Ontologies' hierarchical structure lets developers reuse domain ontologies (for example, of people, devices, and activities) in describing contexts and build a practical context model or compose large-scale context ontology without starting from scratch. And lastly, it provides extensibility. Concepts in the context ontology are organized in form of taxonomies or hierarchies. Newly-defined concept can be easily added into the existing context ontology in a hierarchical manner.

3.1.4 Future Research for Context Ontology

The idea of exploring Web Ontology Language to model context creates opportunities, while also opens up severalopen issues for further study. Here we discuss three main issues: 1) Model Transformation for Machine Learning, 2) Extending OWL for quantitative features, and 3) Privacy Control.

Model Transformation for Machine Learning. As we mentioned, in addition to logic reasoning, machine learning is another feasible approach to derive high-level context from low-level context [35, 36]. In fact, a large amount of context-aware tasks (e.g., a home-care service uses predicted user behaviors to optimize inhabitant comfort) require machine learning mechanisms. Unlike logic reasoning that can be directly supported by ontology models, machine learning requires training data in form of different dedicated models (e.g., Markov chains, feature vectors, etc). Therefore, it requires further study on how to transform ontology based context model to dedicated models for specific learning mechanisms. One direction is to study the model requirements of general machine learning algorithms (e.g., Markov chains, Bayesian learning, neural networks, reinforcement leaning, etc), and provide an algorithm-specific model transformation mechanisms.

Extending OWL for quantitative features. Through the above study in context modeling, we can see that OWL and entailed description logic are necessary for modeling general concepts of context. However, the limitation of description logic makes OWL insufficient for modeling quantitative features of context such as order, quantity, time, quality of information, or uncertainty/probabilities. Unfortunately, capturing such features is critical to certain tasks such as data fusion dealing with uncertain or incomplete sensor context [38]. Therefore, we need study how to extend OWL models with capabilities to express quantitative concepts, thereby enabling temporal reasoning and probabilistic reasoning in a formal approach.

Privacy Control. The dynamism and ubiquity of the pervasive computing paradigm raise important challenges for information security and privacy. Moreover, Semantic Web as a whole is largely conceived as an open network to share information, and can not support any privacy control mechanism. Therefore, we require the context model to provide a working privacy mechanism that can balance knowledge sharing and information privacy for context aware computing.

3.2 Reasoning for High level Context

3.2.1 Context Reasoning Mechanisms.

Different types of entities (software objects) in the environment must be able to reason about uncertainty. These include entities that sense uncertain contexts, entities that infer other uncertain contexts from these basic, sensed/defined contexts, and applications that adapt how they behave on the basis of uncertain contexts. A middleware infrastructure is expected to facilitate computing entities with a variety of reasoning and/or learning mechanisms to help them reason about context appropriately. Using these reasoning or learning mechanisms, entities can infer various properties about the current context, answer logic queries about context or adapt the way they behave in different contexts.

Agents can reason about context using rules written in different types of logic like first order logic, temporal logic, description logic (DL) [26], higher order logic, fuzzy logic, etc. Different agents have different logic requirements. Agents that are concerned with the temporal sequence in which various events occur would need to use some form of temporal logic to express the rules. Agents that need to express generic conditions using existential or universal quantifiers would needto use some form of first order logic (FOL). Agents that need more expressive power (like characterizing the transitive closure of relations) would need higher order logics. Agents that deal with specifying terminological hierarchies may need description logic. Agents that need to handle uncertainties may require some form of fuzzy logic.

Instead of using rules written in some form of logic to reason about context, agents can also use various machine learning techniques to deal with context. Learning techniques that can be used include Bayesian learning, neural networks, reinforcement learning, etc. Depending on the kind of concept to be learned, different learning mechanisms can be used. If an agent wants to learn the appropriate action to perform in different states in an online, interactive manner, it could use reinforcement learning or neural networks. If an agent wants to learn the conditional probabilities of different events, Bayesian learning is appropriate.

3.2.2 Context Reasoning using Description Logic (DL) [26]

Description Logic(DL) allows specifying a terminological hierarchy using a restricted set of FOL (first order logic) formulas. The equivalence of OWL and description logics allows OWL to exploit the considerable existing body of DL reasoning including class consistency and consumption, and other ontological reasoning. The formal semantics entailed by description logic can be used to automatically reason about the context knowledge to fulfill important logical requirements. These requirements include concept satisfiability (whether concept can exist), class subsumption (whether class C is a sub-class of class D), class consistency(whether all the definitions of classes, properties, and relations in a context ontology are satisfiable), and instance checking (whether a context instance is satisfied to the context ontology).

The following is an example for Description Logic Rules to infer user location. If we have the rules that locatedIn has a transitiveProperty, and

isPartOf is subPropertyOf locatedIn,

```
<owl:ObjectProperty rdf:ID="locatedIn">
<rdf:type rdf:resource="&owl;Transitive
 Property" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isPartOf">
<rdfs:subPropertyOf rdf:resource=
 "#locatedIn" />
</owl:ObjectProperty>
```

And if we have the location of a user Bilbo is in bedRoom

```
<Room rdf:ID="bedRoom">
<isPartOf rdf:resource="#home"/>
</Room>
<Agent rdf:ID="Bilbo">
<locatedIn rdf:resource="#bedRoom"/>
</Agent>
```

Then we can can infer bedRoom is in home:

```
<Room rdf:ID="bedRoom">
<locatedIn rdf:resource="#home"/>
</Room>
```

And we can can also infer Bilbo is at home.

```
<Agent rdf:ID="Bilbo">
<locatedIn rdf:resource="#home"/>
</Agent>
```

3.2.3 Context Reasoning using First-Order Logic (FOL)

The user-centric design rationale of context aware systems requires more flexible logic reasoning mechanisms than description logic reasoning. By customizing reasoning rules within the entailment of first-order logic (actually a subset of first order logic, as OWL is less expressive than FOL), a wide range of high-level, conceptual context such as "what the user is doing" can be deduced from low-level context.

The following is a sample rule that infer a user's likely situation based on context, activity, location, and computing entity.

```
type(?user, User), type(?event, Meeting),
location(?event, ?room), locatedIn(?user,
?room),
startDateTime(?event, ?t1),
endDateTime(?event, ?t2), lessThan(?t1,
currentDateTime()),
greaterThan(?t2, currentDateTime())
=>situation(?user, AtMeeting)
```

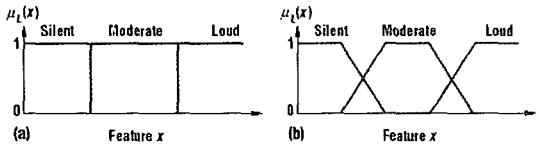
3.2.4 Context Reasoning using Probabilistic and Fuzzy Logic

Probabilistic logic lets us write rules that reason about events'probabilities in terms of the probabilities of other related events. An example rule (written in XSB) is

```
prob(X, Y, union, P) :- prob(X, Q), prob(Y,
R), disjoint(X, Y), (P is Q + R).
```

This rule essentially says that Pr(X U Y) = Pr(X) + Pr(Y) if X and Y are disjoint events, that is, they never occur together. X and Y can be context predicates. For example, X could be location(Bob, in, Room 2401) and Y could be location(Bob, in, Room 3234). X and Y are disjoint events, since Bob can't be in two different locations at the same time [15].

Fuzzy logic is somewhat similar to probabilistic logic. In fuzzy logic, confidence values represent degrees of membership rather than probability. Fuzzy logic is useful in capturing and representing imprecise notions such as "tall," "trustworthy," and "confidence"and reasoning about them.

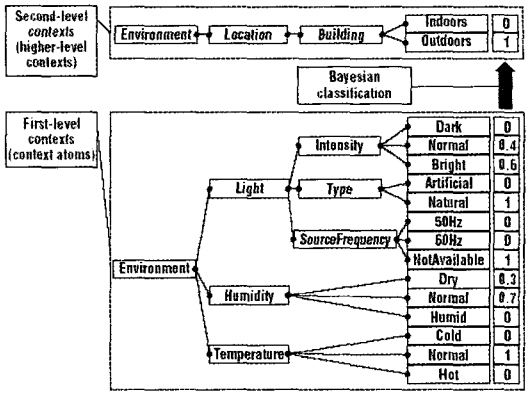


(Figure 3) An example of Fuzzy Logic for audio feature [27]

For example, in quantizing environment sound intensity, the quantization divides the processed feature into three quantities - Silent, Moderate, and Loud - corresponding to the three membership functions denoted as L(x). Apply a fuzzy set for features, resulting in continuous valued fuzzy labeling (Figure 3). For example, L(x) = 0.7 / Silent + 0.3 / Moderate + 0 / Loud [27].

3.2.5 Context Reasoning using Bayesian Networks

Bayesian networks are directed acyclic graphs, where the nodes are random variables representing various events and the arcs between nodes represent causal relationships. Bayesian networks are a powerful way of handling uncertainty in context data, especially when there are causal relationships between various events. They are useful for performing probabilistic sensor fusion and higher-level context derivation. In general, root nodes in the Bayesian network represent the information to be deduced, while the leaves are sensed information. The intermediate nodes are important sub-goals that are helpful to the deduction process.



(Figure 4) An example of Bayesian Network to deduce the location (indoor/outdoor)

Figure 4 introduces the formation of the higher level context Outdoors from the context atoms (or low level sensed contexts) using a nave Bayesian network [27]. White rectangular boxes represent types and light tan boxes represent context values. Dark tan boxes contain the corresponding confidence instance values for the current situation. The nave Bayesian network classifies the confidence instance values into one of the previously defined output classes, Indoors and Outdoors in this network.

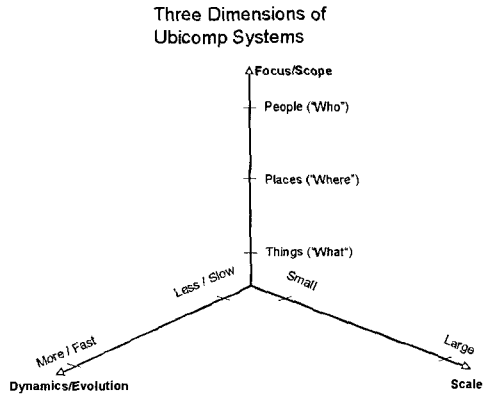
The nave Bayes classifier works well for online context inference and sensor fusion mainly because 1) It has proven robust even with missing, uncertain, and incomplete information.5. 2) It is computationally efficient. Training and inference both have a linear complexity in input data size. And 3) It requires no background information modeling except for choosing the relevant network inputs.

3.2.6 Challanges in Context Reasoning

The decision on what kind oflogic or learning mechanism to use depends not only on the power and expressivity of the logic, but also on other issues like performance, tractability and decidability. According to the feature of the tasks, different learning mechanisms can be used in ahybrid manner. For example, learning based on Bayesian networks and explicit rules written in probabilistic or fuzzy logic are useful in different scenarios. Bayesian networks are useful for learning the probability distributions of events and enable reasoning about causal relationships between observations and the system state. They, however, must be trained before they can be used, but because they are flexible and can be retrained easily, they can adapt to changing circumstances. Probabilistic logic is useful when we have precise knowledge of events'probabilities: fuzzy logic is useful when we want to represent imprecise notions. Both probabilistic and fuzzy logic are useful in scenarios where getting data to train a Bayesian network is difficult. This is especially true in the area of security. Beside this first challenge in choosing the most relevant reasoning mechanism for each high level context, middleware system must also facilitate the ability to plug in new reasoning mechanisms. The use of a fixed APIs between the reasoning engines and other software entities using them appears to be a feasible solution for adding different reasoning engines easily.

3.3 Consideration Factors for System Paradigms

Let us have a look at some of the envisioned consequences for research in applications of Ubiquitous Computing systems. In this section we try to give taxonomy of typical Ubicomp applications. The relevant coordinates for the taxonomy are depicted in Figure 5, and we believe that these scales are important for characterizing Ubicomp applications on a qualitative level, which in turn affect the middleware system paradigm.



(Figure 5) Three Dimensions of Ubicomp Systems

3.3.1 System Focus or Scope

Applications are designed to cover some application domain. A commonclassification in this area is "People, Places, and Things" originating from the HP Cooltown research project [28]. Usually, one can identify a central focus of an application related to one or two of these topics.For example, a tourist guide is focused on giving information to a user. This is the main focus. However, it gives information about places: therefore, it has places as a secondary focus. But to be successful it has to bring information about places to users in an appropriate way (for the user), hence, "people" is the main focus ("subject") and places are the secondary focus

("objects") of the system.

3.3.2 System Scale

Scale is an important factor for system and application design. Is a system designed for a rather small place and covers a rather small range (like, e.g., a Smart Home application) or does it scale up to something like a public space (e.g., a train station) or even to cover something larger like a city or a country (geographical scale)? Obviously, this has some fundamental influence on system design. Another sub-dimension is the number of participants in a system. A Smart Home application might have to handle a couple of dozens objects and subjects (the fridge, the home entertainment center, the garage door, etc. Here the requirements for system design are completely different. For example for a smaller application it might be perfectly sufficient to have no common infrastructure for data exchange and communication, and an ad-hoc network and broad- or multicast communication might suffice. On a larger scale this approach is not feasible due to the traffic this style of communication implies.

3.3.3 System Dynamics/Evolution

This is the most driving factor for Ubicomp applications. Ubicomp is inherently "mobility driven" and "dynamic": People are moving, as well as things, and sometimes even the places are moving around (e.g., cars or trains). This imposes serious design challenges for system, as well as for application designers: Associations between communicating parties are volatile, so might be data/information (especially by third parties), data is belonging to someone else: everything is - in a sense - "floating around." And the consequences would be:

- System and application design will have to take dynamics (on every level) into account
- Bindings will be casual and volatile, due to mobility.
- Systems have to be built along with a changing society, not against.

Also a consequence of the highly dynamic nature of Ubicomp applications is the need for evolvable systems. It can be expected that such systems must be open, flexible, and extensible. The reason is obvious: we cannot make assumptions about the capabilities, the operating systems, the installed software, or the style of communication devices have. On the other hand, we cannot make assumptions about what services can be used, what QoS can be expected, or what "version" of service is "installed". Therefore, it is mandatory to build systems in an open, flexible, and extensible fashion. Especially in highly dynamic systems with many different users this requirement is extremely important. Systems might be more long-lived than devices the users bring along: therefore, they have to be extensible to new standards and requirements. On the other hand, users cannot make assumptions about the services and infrastructure they will find at different places. Their devices must be capable of adapting to the situation found at hand. For instance, service discovery is important. Not every public place has the same infrastructure and services "installed".party has to have the capability to evolve and adapt.

A mandatory requirement for Ubicomp systems and applications is that the actual functionality is

determined at "runtime" opposed to determine the functionality at "compile time," i.e., the moment the device is deployed. Here an important requirement is to have an effective and efficient data-management in term of retrieval of relevant information, like relevant services, relevant data (e.g., for Context determination) and other information needed depending on the actual application.

4. Summary

In this article we address some system characteristics and challenging issues in developing Context-aware Middleware for Ubiquitous Computing. The functionalities of aContext-aware Middleware includes gathering context data from hardware/software sensors, reasoning and inferring high-level context data, and disseminating/delivering appropriate context data to interested applications/services. The Middleware should facilitate the query, aggregation, and discovery for the contexts, as well as facilities to specify their privacy policy. Following a formal context model using ontology would enable syntactic and semantic interoperability, and knowledge sharing between different domains. Middleware should also provide different kinds of context classification mechanisms as pluggable modules, 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 mechanism that best meets the reasoning requirements of each context. And finally, to promote the context-trigger actions in application level, it is important to provide a uniform and platform-independent interface for applications to express their need for different context data without knowing how that data is acquired. The action could involve adapting to the new environment, notifying the user, communicating with another device to exchange information, or performing any other task.

References

[1] M. Weiser: Scientifc America. The Computer for the 21st Century. (Sept. 1991) 94-104: reprinted in IEEE Pervasive Computing. (Jan.-Mar. 2002) 19-25

[2] http://www.ubiq.com/hypertext/weiser/SciAm Draft3.html

[3] Schilit, B. N., N.I. Adams, and R. Want: Context -Aware System Architecture for Mobile Distributed Computing. PhD thesis, 1995.

[4] 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)

[5] Pascoe J., Ryan, N. S. & Morse, D. R. (1998). Human-Computer-Giraffe Interaction HCI in the field. Proceedings of the Workshop on

Human Computer Interaction with MobileDevices, Glasgow, Scotland.

[6] Context Toolkit project http://www.cs.berkeley.edu/~dey/context.html

[7] W. Emmerich. Engineering Distributed Objects. John Wiley and Sons, Ltd., 2000.

[8] K. Edwards. Core JINI. Prentice Hall, 1999.

[9] Sun. Javaspaces. http://www.sun.com/jini/specs/jini1.1html/js-title.html, 2001.

[10] IBM. T Spaces. http://www.almaden.ibm.com/cs/TSpaces/, 2001.

[11] Xiaodong Jiang, James A. Landay: Modeling Privacy Control in Context Aware Systems. IEEE Pervasive Computing, Vol. 1, No. 3 July-September 2002.

[12] Philip Gray and Daniel Salber: Modeling and Using Sensed Information in the Design of Interactive Applications. 8th IFIP International Conference, EHCI 2001, Toronto, Canada, 2001

[13] Guanling Chen and David Kotz: Context Aggregation and Dissemination in Ubiquitous Computing Systems. In Proceedings of the Fourth 1EEE Workshop on Mobile Computing Systems and Applications, June 2002.

[14] Jeffrey Heer,Alan Newberger, Chris Beckmann, and Jason I. Hong: liquid: Context -Aware Distributed Queries. Ubiquitous Computing, 5th International Conference, Seattle (UbiComp 2003), October 12-15, 2003.

[15] Anand Ranganathan, Roy H. Campbell: A Middleware for Context -Aware Agents in Ubiquitous Computing Environments. In ACM/IFIP/USENIX International Middleware Conference, Brazil, June, 2003.

[16] S. K. Das, D. J. Cook, A. Bhattacharya, E. O. Heierman, III, and T.-Y. Lin: The Role of Prediction Algorithms in the MavHome Smart Home Architecture, IEEE Wireless Communications Special Issue on Smart Homes, 9(6), pages 77-84, 2002.

[17] G. Thomson, S. Terzis, and P. A. Nixon: Towards Dynamic Context Discovery and Composition. The First UK-UbiNet Workshop, London, UK, 2003.

[18] Guanling Chen and David Kotz.: Solar: An Open Platform for Context Aware Mobile Applications. In Proceedings of the First International Conference on Pervasive Computing (Pervasive 2002), Switzerland, June, 2002.

[19] Hong, J. I., et al.: An Infrastructure Approach to Context -Aware Computing. HCI Journal, 2001, Vol. 16.

[20] Asim Smailagic, Daniel P. Siewiorek, Joshua Anhalt, David Kogan, and Yang Wang, : Location Sensing and Privacy in a Context Aware Computing Environment. Pervasive Computing, 2001

[21] S. S. Yau, F. Karim, Y. Wang, B. Wang, S.Gupta: Reconfigurable Context-Sensitive Middleware for Pervasive Computing. (Jul.-Sep. 2002) 33-40

[22] J. Davies, D. Fensel, F. V. Harmelen: Towards the Semantic Web, Ontology-Driven Knowledge Management, John Wiley & Sons. (Nov. 2002)

[23] W3C Web Ontology Working Group: The

Web Ontology language: OWL. http://www.w3.org/2001/sw/WebOnt/

[24] Klyne, G., Caroll, J. J.: Resource Description Framework Abstract Concept and Syntax. W3C Recommendation. (10 Feb. 2004)

[25] Xiaohang Wang, Daqing Zhang, Jinsong Dong, et al., Semantic Space: A Semantic Web Infrastructure for Smart Spaces. IEEE Pervasive Computing, Vol. 3 No. 3, 2004

[26] Baadar, F., Horrocks, I., Sattler, U.: Description Logics. Handbook on Ontologies. (2004)3-28

[27] Korpipaa, P.: Mantyjarvi, J.: Kela, J.: Keranen, H.: Malm, E.J. Managing context information in mobile devices.: Pervasive Computing, IEEE ,2 ,3 ,2003 Pages:42 - 51

[28] Cooltown Project, http://www.cooltown.com /cooltown/index.asp

The Authors

Hung Q. Ngo

Hung Q. Ngo is a Master student in the Computer Engineering Department at Kyung Hee University. His research interests include Real-time Embedded Systems, Middleware for Context-awareness, Middleware for Sensor Network, and Ubiquitous Computing. He received a BS in Electrical and Electronic Engineering from HoChiMinh City University of Technology (HUT), Vietnam, with honors. Contact him at nqhung@oslab.khu.ac.kr

Sungyoung Lee

Sungyoung Lee is a Professor in the Department of Computer Engineering, Kyung Hee University, Korea. His research interests include Ubiquitous Computing, Middleware, Embedded System, and Mobile Computing. He received a PhD in Computer Science from Illinois Institute of Technology, Illinois, USA. He is a member of the IEEE, the ACM, the KISS and the KIPS. Contact him at sylee@oslab.kyunghee.ac.kr.

Acknowledgements

This works is supported in part by the Ministry of Information and Communications'ITRC Program (joint with Sun Moon University)