The context-aware middleware in ambient intelligence

Tao Xu

ÉCOLE CENTRALE LYON

N° d'ordre : 2013-47

Thèse de l'Université de Lyon

The Context-aware Middleware in Ambient Intelligence

Pour l'obtention du titre de
Docteur de l'Ecole Centrale de Lyon

Ecole Doctorale
Informatique et Mathématiques

Spécialité
Informatique

Par
XU Tao

Soutenue publiquement le 9 décembre 2013, devant la commission d'examen :

Christophe KOLSKI	Professeur – Université de Valenciennes	Rapporteur
Serge GARLATTI	Professeur – Télécom Bretagne	Rapporteur
Bertrand DAVID	Professeur – Ecole Centrale de Lyon	Directeur
René CHALON	Maitre de Conférences – Ecole Centrale de Lyon	Co-directeur
Jean-Paul BARTHES	Professeur Emérite – U.T.C. Compiègne	Président
Chuantao YIN	Assistant Professor – Ecole Centrale de Pékin	Examinateur

UNIVERSITÉ DE LYON

Acknowledgements

PhD for me was once a distant dream, and the studying processing was full of hardships. Such a daunting task, if not many people guidance, support and assistance, is impossible to successfully complete.

I owe my supervisor, Prof. Bertrand David, my eternal gratitude. He has advised, encouraged, and inspired me through the arduous doctoral process. He provided the precious opportunity to continue my PhD in France when I was helpless. He introduced me to the field of Ubiquitous computing, guided me throughout my graduate work. He always believed in me and support me do research in the field that I was interested in. I feel extremely lucky to have worked under his guidance.

I am also indebted to my co-supervisor, Prof. René Chalon who has been patient, helpful, and supportive during my research. He created a good environment to study and constantly motivated and encouraged me to perform high quality research.

I'd like to thank the members of Silex group in Ecole Centrale de Lyon, including Huiliang Jin, Bingxue Zhang, Chen Wang, and Florent Delomier for all they helped me in life and research. Boyang Gao, Huibing Li, Taiping Zhang, Chao Zhu, Zhe Li and all my friends in France, helped make my time especially enjoyable and never feel alone even so far away from my mother land.

I also gratefully acknowledge the financial support of China Scholarship Council, which provide a great chance for me to study aboard.

I would like to thank my family who have inspired and supported me throughout my life. I am very grateful to my parents for their constant encouragement and support. They bestowed me a love for learning, and showed me the way. Finally, I would like to thank my girlfriend Yun Zhou, (she has become my wife now), who help me greatly in both my life and research, and has always supported me emotionally throughout my doctoral studies.

Table des matières

1 Introduction

1.1 Ubiquitous Computing

> *"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it."*
>
> *Mark Weiser*

Almost 20 years ago, Marc Weiser viewed the prospect of computing in the 21st century, and proposed the pioneering notion of ubiquitous computing, also known as pervasive computing. The essence of Weiser's vision is that mobile and embedded processors can communicate with each other and the surrounding infrastructure, seamlessly coordinating their operation to provide support for a wide variety of everyday work practices (Krumm, 2009).

When we review the history of computer developments, we find that it encompasses three eras. The first era is the era of mainframe computers beginning in the 1950s. Few users share one mainframe computer at the same time. The second era is the era of personal computers. The emergence of the first consumer computers (Scelbi & Mark-8 Altair & IBM 5100) marked a personal computer revolution throughout the world. Due to the rapid development of hardware and software technologies and the sharp decline in size and price of computers, use of personal computers became widespread. One user owns and uses one personal computer.

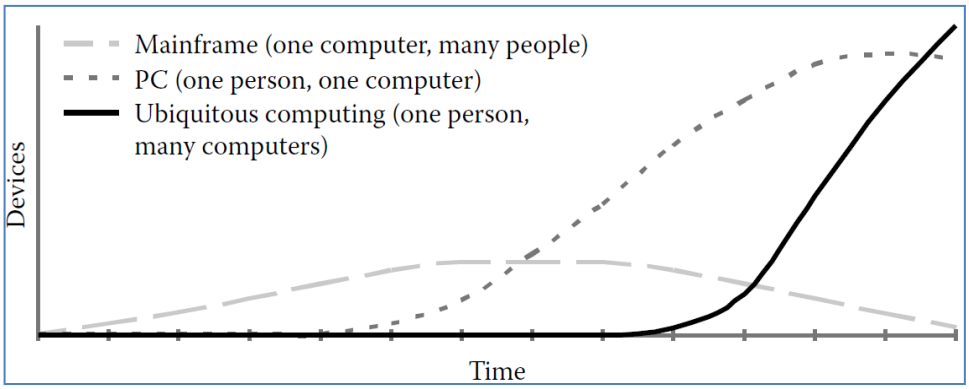


Figure 1.1 Three computer development eras

Ubiquitous computing is considered as the third era following on from mainframe computers and personal computers. According to Moore's Law: "the computer power available on a chip doubles every two years", chips will become much smaller and cheaper in the future. Chips can be embedded in everything, not only computers, smartphones and tablets, but also in objects of daily use such as glasses and watches. The numerous computing devices become integrated into everyday life. Thus one user owns and uses many computers (computing devices).

Ubiquitous computing (often abbreviated to "ubicomp") refers to the concept of "every-where" computing. It aims at a heterogeneous set of devices, including invisible computers embedded in everyday objects such as cars and furniture, mobile devices such as personal digital assistants (PDAs) and smart phones, personal devices such as laptops, and very large devices such as wall-sized displays and tabletop computers situated in the environments and buildings we inhabit (Krumm, 2009). On the other hand, ubiquitous computing strives to make computing and communication transparent to users. Thus, Weiser also viewed ubiquitous computing as "The coming age of calm technology".

Because of its attractive prospect, Ubiquitous Computing drew the attention of the science community right from the start. According to statistics, almost all prestigious universities conducted research on Ubiquitous Computing.

- CMU proposed a ubiquitous computing project Aura (Sousa et al., 2002) providing each user with an invisible halo of computing and information services that persists regardless of location. This focuses on the design and development of pervasive computing middleware and its applications.

- UC Berkeley developed Dust Networks and Tiny OS (Hill et al., 2000). The former attempted to demonstrate that a complete sensor/communication system could be made of sensors one cubic millimeter in size. This involved progress in miniaturization, integration, and energy management: the latter is an open-source operating system optimized for sensor networks, which is widely used in sensor network research.

- MIT's Oxygen project (Rudolph, 2001) involved a full range of software, hardware and networks in ubiquitous computing. Their goal is that computing will be freely available everywhere, just like oxygen in the air we breathe.

- The Portolano project (Esler et al., 1999) was an initiative of the University of Washington. It established a test bed for investigation into the emerging field of ubiquitous computing. Its goal is In-visible computing, a term invented by Donald Norman to describe the coming age of ubiquitous task-specific computing devices. Researchers attempt to infer user intentions based on the behavior and habits of their daily life, and provide customer satisfaction services.

- The Interactive Workspace project (Johanson et al., 2002) was started at Stanford University. It studied intelligent space and aimed at investigating the design and use of rooms containing one or more large displays with the ability to integrate portable devices and to create applications integrating the use of multiple devices in space.

- UIUC's contribution to ubiquitous computing was Gaia (Román et al., 2002), a distributed middleware infrastructure that coordinates software entities and heterogeneous networked devices contained in a physical space. Gaia exports services to query and utilize existing resources, and access and use current context, and provides a framework to develop user-centric, resource-aware, multi-device, context-sensitive, and mobile applications.

Besides the United States, the European Union also gave financial support to a series of related research on ubiquitous computing, such as the Disappearing Computer Program, TEA plans.

In the early days of ubiquitous computing research (circa 1990), finding an appropriate journal to publish papers and discuss research was challenging. In fact, no conferences at that time were at all suited to the topic (Krumm, 2009). At present, the research community warmly embraces ubiquitous computing. For papers more suitable to journals and maga-

zines, a number of peer reviewed publications have also been created over the years, such as “Personal and Ubiquitous Computing” founded in 1997, “IEEE Pervasive Computing” founded in 2001, and “IEEE Transactions on Mobile Computing” founded in 2002. In contrast to the early 1990s, from 2009 onward there is a wealth of ubiquitous computing related conferences available to researchers. Some have grown into the top-tier conferences in this area, representing the highest level in ubiquitous computing, like UbiComp (Intemational Conference on Ubiquitous Computing) founded in 1999, Pervasive (International Conference on Pervasive Computing) founded in 2000, and PerCom (IEEE International Conference on Pervasive Computing and Communications) founded in 2003.

Currently, Ubiquitous Computing is seen to comprise any number of mobile, wearable, distributed and context-aware computing applications (Pallapa, 2010).

Many aspects of Weiser’s visions have already become reality in the past two decades. Furthermore, one of his primary ideas has recently evolved into a more general paradigm known as context awareness. Detailed information about context-aware computing is given in the following section.

1.2 Context-aware Computing

Today, context-aware computing has become a central research theme in many other ubiquitous computing programs. It requires that systems adapt their functionality to a user’s activity and situation in the environment. (Satyanarayanan, 2001) and (Strang and Linnhoff-Popien, 2004) have concluded the flow of computing evolution based on previous achievements, as shown in Figure 1.2. They viewed context awareness as the core attribute of ubiquitous computing.

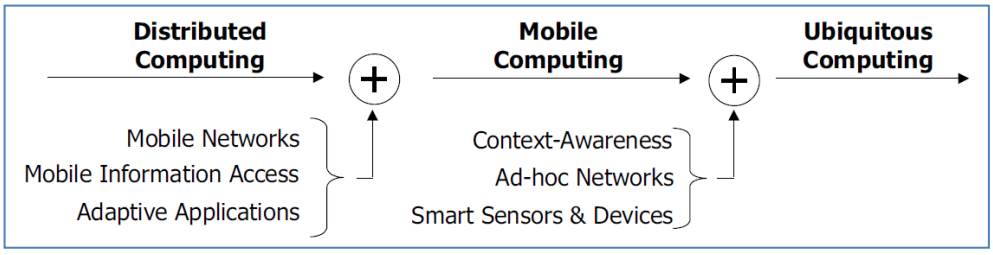


Figure 1.2 the flow of computing evolution

The first context-aware system, Active Badge System (Want et al., 1992), emerged in 1992. Since then, hundreds of context-aware systems and context-aware applications have been proposed. Context-aware computing, as the core research theme in ubiquitous computing, is increasingly attracting the attention of researchers.

1.2.1 Definition of context

Understanding "What's context?" is the first step toward using context efficiently. The Oxford English Dictionary provides a generic definition of context in English:

> *"The circumstances that form the setting for an event, statement, or idea, and in terms of which it can be fully understood."*

Context in ubiquitous computing is subtly different from that in English. In literature, the term "context" first appeared in Schilit and Theimer's paper (Schilit and Theimer, 1994). They defined context as location, identities of nearby people, and objects and changes to those objects. In addition, they divided context into three categories: Computing context, User context, Physical context. Since then, many researchers have made efforts for defining context in Ubiquitous Computing. Brown et al. (Brown et al., 1997) give context definition as location, identities of the people around the user, the time of day, season, temperature, etc. Context as defined by Ryan et al. (Ryan et al., 1998) is the user's location, environment, identity and time. Another way of defining context is to use its synonyms. The definition of context given by Franklin and Flaschbart (Franklin and Flachsbart, 1998) is the situation of the user. Similarly, Hull et al. (Hull et al., 1997) describe context as the aspects of the current situation. These kinds of definitions are vague and limited. Finally, Day and Abowd (Dey and Abowd, 1999) summarize previous works and provide a definition of context more generally with the following statement:

> *"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves."*

This definition of context is generally accepted by researchers. In this thesis, we also adopt this definition to define "context". Making sense of the definition of context means just beginning to study context-aware compu-

ting. In the following subsection, we will introduce another crucial conception in this research field.

1.2.2 What is context awareness?

As described earlier, the Active Badge System of 1992 is commonly considered to be the first research investigation into context awareness. However, the first definition of the term "context-aware" was given by Schilit and Theimer two years later. They (Schilit and Theimer, 1994) described context awareness as follows:

> *"It adapts according to its location of use, the collection of nearby people and objects, as well as changes to those objects over time."*

From then on, there have been numerous attempts to define context awareness.

Hull et al.(Hull et al., 1997) and Pascoe et al. (Pascoe et al., 1998) have described context-aware computing to be the ability of computing devices to detect and sense, interpret and respond to aspects of a user's local environment and the computing devices themselves. Ryan (Ryan, 1997) viewed context awareness as applications that monitor input from environmental sensors and allow users to select from a range of physical and logical contexts according to their current interests or activities. In a similar definition, Fickas et al. (Fickas et al., 1997) gave context-aware application (known as environment-directed application) a definition that the application can monitor changes in the environment and adapt their operation according to predefined or user-defined guidelines. Context-aware application defined by Brown (Brown, 1998) was the application that automatically provides information and/or takes actions according to the user's present context as detected by sensors. Until 1999, Day and Abowd (Dey and Abowd, 1999) have provided a widely acceptable definition of context awareness:

> *"A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task."*

The precise definition of context awareness is an efficient method for identifying whether or not an application is context-aware, and has pro-

vided researchers with a useful way of determining what types of applications to focus on.

1.3 Challenges in Context-aware Computing

Context-aware applications are becoming increasingly more prevalent and can be found in the areas of wearable computing, mobile computing, robotics, adaptive and intelligent user interfaces, augmented reality, adaptive computing, intelligent environments and context-sensitive interfaces (Krumm, 2009). Given the focus on building context-aware applications, it is important to understand what the main issues are in this research theme. How to acquire context, how to perceive context and how to create context-aware applications are all issues still faced with enormous challenges in both research and practice. We will discuss four major challenges that researchers need to consider when conducting research in context-aware computing.

- **Context Sensing**

  This is the first process for all context-aware applications. It refers to an ability to detect contextual information from the diverse context sources, which not only contains heterogeneous physical sensors/devices, but also proliferating virtual web sources. The process includes understanding what type of contextual information the sensors provide, communicating with context sources by diverse communication protocols: HTTP, REST, SOAP etc. The extracted raw data will be dispatched to different parts for further processing at a later stage. In ubiquitous computing environments, there are a variety of kinds of context with different data formats. Context acquirement and context integration face many difficulties.

- **Context Discovery**

  Context discovery allows context-aware applications to locate and exploit resources and services that are relevant to the user's context. The context can be extracted by different context sources according to context sensing. Context Discovery is responsible for providing the appropriate context source for different context-aware applications

- **Context Modeling**

Context Modeling refers to defining and storing context data in a machine processable form (Baldauf et al., 2007). As an important fundamental process, it lays the ground work for the whole context-aware system. Raw context information is obtained from various heterogeneous context sources, and has its own expressions and models. It cannot be directly employed by context-aware applications. The context model is in charge of presenting and organizing context, and should provide a standardized model for facilitating the other component of context-aware systems to access and handle context. To date there is no uniform standard for context modeling, and it is difficult to exchange context information in context-aware systems. There are six main methods for constructing context models: Key Value Models, Markup Scheme Models, Graphical Models, Object Oriented Models, Logic Based Models, and Ontology Based Models. The detailed description of these will be presented in the following chapter. In context-aware computing, it is very important to know how to use a wide range of heterogeneous context information to establish a unified abstract logical model that can ensure context is easily expressed and shared.

- **Context Interpretation**

Context Interpretation is related to context management and context distribution. It has three main tasks: context aggregation, context query, and context inference.

- *Context aggregation:*

Context is retrieved from the different context resources. The context-aware system conceals the complexity of physical sensors or other context resources by providing a high level of abstraction enabling context resource independence. This should be a reusable architecture component, simplifying integration and minimizing development efforts.

- *Context query:*

This is a query and notification mechanism, helping to find appropriate context and returning the results to requesting clients. Context is different from conventional data, and represents the situation of objects containing not only data but also relationships between data. For example, the context-aware application wants to know “who is the computer science professor in Room 3”. So, the context query is necessary for looking for context with

its relationships. It needs high-level mechanisms or solutions to deal with this issue, such as context query language and query optimization.

- *Context Inference:*

As the crucial part in context interpretation, its role is like a brain. Currently, it has two main tasks: checking context consistency and using inferring/learning techniques to deduce high-level, implicit context requested by context-aware intelligence services from related, low-level, explicit context. High-level context is defined as the context that cannot be directly extracted from the context source: physical sensors or virtual sources. The solution for context consistency depends on the chosen context model. Inferring high-level context (mainly refers to user's activity) is a complex problem. Many researchers have attempted to apply numerous methods to solve this problem. According to their principles, these methods can be grouped into two categories: Rules, and Machine learning algorithms. Rules can infer user's activity based on a set of first order logic rules. If low-level context satisfies the condition, then high-level context (user's activity) will be determined. The machine learning algorithm learns the model parameters of user's activity from the historical context data and uses this model to recognize user's activity. The detailed information will be presented in chapter 5. An intelligent support for context inference will simplify implementation of the context-aware application.

1.4 Context-aware System

Researchers propose different architectures for implementing Context-aware systems based on the specific requirements and conditions, such as sensor position (local or remote), the number of possible users (one or more users), the available device or the ability for system extension. When designing context-aware systems, researchers always consider first the methods for accessing context data. In other words, these methods are predefined system architecture types to some extent. In the existing studies, there are two development stages for context-aware computing system architectures:

- **Context-aware system based on directly accessing the sensors**

In the early days, context-aware systems were developed generally for a specific application. These context-aware applications usually interacted with sensors and hardware devices directly, and these applications and con-

texts were close-coupled. To some extent, this method improved system efficiency, but it limits system scalability, weakens adaptability to the environment, and reduces system reusability.

- **Middleware for Context-aware System**

  In ubiquitous computing environments, systems must perceive dynamic and changing context information and take action accordingly. Therefore, to meet these requirements, it is necessary to separate context awareness and application development, that is, to separate the underlying sensing data processing and high-level context-aware applications. What is needed is an intermediate layer, namely context-aware middleware. Context-aware middleware introduces a hierarchical structure, used to hide the underlying sensing detail, and to provide the necessary context-aware services to applications, such as: collecting raw data from a variety of context resources, converting heterogeneous data to a uniform format, deducing high-level context and delivering appropriate context to the required applications. Compared with the approach directly accessing the sensors, middleware can easily be extended. Since it is not necessary to modify the application code, it also enhances hardware independence and improves system reusability. In contrast to client-server architecture, middleware effectively hides heterogeneous underlying hardware and complex network communications. As it enhances system portability and scalability, it provides more direct and effective support to context-aware application developers. Middleware, as an efficient and practical structure, is widely applied in Context-aware computing.

1.5 Research Contributions

Context-aware computing, as the central research theme in ubiquitous computing, is faced with enormous challenges from theory through to practice. In this dissertation, we review the research path on context-aware computing, summarize the previous related work, and discuss the main challenges in this field. Then we investigate in-depth some chosen key issues in context awareness and develop a context-aware middleware. The main research contributions are presented in three categories: a spatial-temporal context representation model, a context-aware middleware, and an intelligence context inference engine.

- **A spatial-temporal context representation model**

  The spatial-temporal context representation model is proposed to organize context and relationships for the context-aware system. Context modeling is the specification of all entities and relationships between these entities, needed to describe the context as a whole. We apply the ontology-based method to construct our model, supporting both knowledge sharing and reuse as well as logic inference. This model adopts a two-layer hierarchy structure for different situations. The higher layer covers the generic common context, while the lower layer focuses on various specific situations. Differing from existing models, besides taking location factors into account, it supports different historical context services depending on varying context resources. These historical contexts may be used to predict and infer context.

- **A context-aware middleware**

  A context-aware middleware is designed as a platform associated with service discovery, mobility, environmental changes, and context retrieval. It is organized in two layers: the low layer is a sort of Enterprise Service Bus, while the high layer is the versatile context interpreter. The low layer provides a solution for integrating sensors and actuators with a standardized data representation and unified standard interface to achieve the core functions of service interaction: service registry, service discovery, and service consumption. The versatile context interpreter is in charge of context inferences, expressive query, and persistent storage. It consists of four parts:

  - The context aggregator is responsible for working with basic contextual data collected by the low layer.

  - The context knowledge base provides persistent storage for context through the use of relational databases, as well as supplying a set of library procedures for other components to query and modify context knowledge.

  - The context query engine has two main tasks: one is to handle queries from the application. It supports SPARQL, which is an RDF query language, able to retrieve and manipulate data stored in OWL. The other is to invoke the context inference engine. When the application needs high-level context, it will invoke the context inference engine to generate the inferred context.

- The intelligent inferent engine concerns main research contributions. The details will be provided in the next section.

- **An intelligent inference engine.**

  The intelligent inference engine is the central and intellectual component of context-aware middleware. We review all the methods concerning activity context recognition published in three premier conferences in the past decades and conclude that activity context recognition is divided into three facets: basic activity inference, dynamic activity analysis, and future activity recommendation. Then we propose an intelligent inference engine based on our context-aware middleware. This integrates the three most popular methods of activity context recognition used in context-aware application, and provides a solution for satisfying different requirements in activity context recognition.

1.6 Organization of the dissertation

In this dissertation, we investigate a context-aware middleware integrating heterogeneous sensors and various technologies to simplify the context-aware application implementation process. The remainder of the dissertation is structured as follows:

- In chapter 2, we review the state of the art and related research relevant to context-aware computing. An in-depth discussion of existing context-aware systems is presented. Following these discussions, these systems are analyzed and compared.

- In chapter 3, we outline the main methods for constructing the context representation model, and then propose our context model, which uses ontology to organize context in a spatial-temporal manner and paves the way for our context-aware middleware.

- In chapter 4, a context-aware middleware is proposed to provide an efficient platform for rapid prototyping of context-aware services in ubiquitous computing environments. The infrastructure of context-aware middleware is presented, along with a concise description of each component. Following these descriptions, we use a scenario to interpret its work flow.

- In chapter 5, we propose an intelligent inference engine for context-aware middleware, made up of a basic inference module and an intelligent inference module. Besides satisfying requirements for checking context consistency, our inference engine integrates the three most popular methods for activity context recognition: Rules, Decision Tree, and Hide Markov Model. It provides a solution for all facets of activity context recognition based on our context-aware middleware.

- In chapter 6, two scenarios are designed and implemented to explain how to use our context-aware middleware to build context-aware applications. We analyze and compare our middleware with five representative context-aware systems in five aspects at the end of chapter.

- The conclusion and future works are provided in chapter 7. In this chapter, we conclude as to our main contributions, and discuss our future directions.

2 State of the Art in Context-aware Computing

2.1 Introduction

The Holy Grail of Ubiquitous Computing is to seek a manner that enables humans to interact with computers naturally. Context-aware computing provides an approach for using context as an implicit cue to enrich the impoverished interaction between humans and computers, making it easier to interact with computers (Krumm, 2009). More than 20 years have elapsed since Roy Want et al. developed the Active Badge System (Want et al., 1992), generally considered as the first context-aware application. Today, context-aware computing has become a prosperous emerging field and draws increasing attention from researchers.

In this chapter, at first five representative existing context-aware systems are reviewed. Then we analyze and compare those based on five key features in context-aware computing: architecture, context representation, context sensing, context storage, and context reasoning. Finally, the chapter ends with a summary and conclusion.

2.2 Context-aware System

The context-aware system, as the crucial component in Ubiquitous Computing, perceives context changes and adapts accordingly. It provides an important way to improve system intelligence, and comes up with a convenient and efficient interaction method between the intelligence system and the surrounding environment. The context-aware system has a general architecture divided into three tiers: context-aware application, context-aware service, and context sensors.

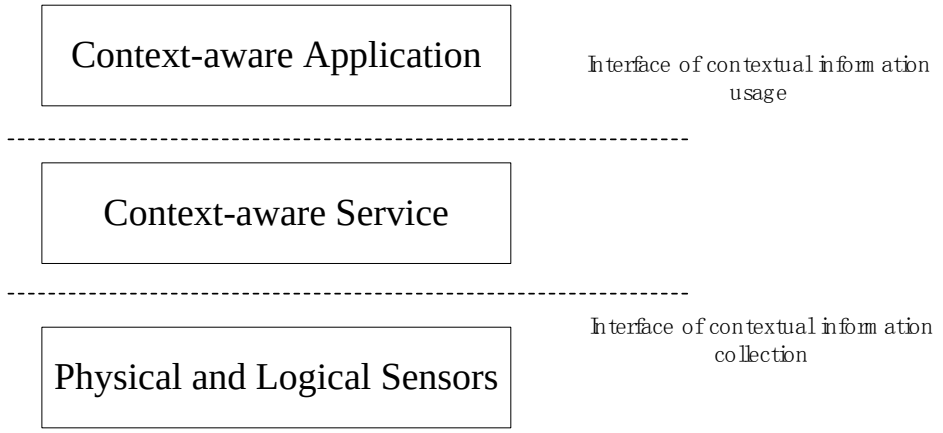


Figure.2.1. The general architecture for the context-aware system

The context sensors tier refers to generalized sensors containing physical devices or software. Amongst these tiers, the context-aware service tier separates context sensing from application development, providing a universal interface for the higher level to develop context-aware application and support to access diverse sensors in the lower level. This separation simplifies context information handling and management. The context-aware service structure is often subject to a number of factors, such as the user device, whether to support multi-users, or system scalability. Architecture design will vary according to application requirements. We will outline and analyze the five representative existing context-aware systems below.

2.2.1 Context Toolkit

The Context Toolkit (Dey, 2000), developed by Anind K. Dey, is a toolkit supporting development of context-aware applications. It provides a platform to obtain context-sensitive information and send context information

to context-aware applications. In addition, the Context Toolkit can be used as a research test platform, supporting more complex studies of context awareness, such as the development of high-level abstract programming, processing of unconcerned context information, and accessing control of user's context information.

Context Toolkit architecture is shown below. It consists of three main components: widget, aggregator and interpreter. Context widgets are charged with retrieving context from sensors. Context aggregators combine many widgets to collect the entire context concerning a given entity. Interpreters provide high-level context to applications based on low-level context from widgets.

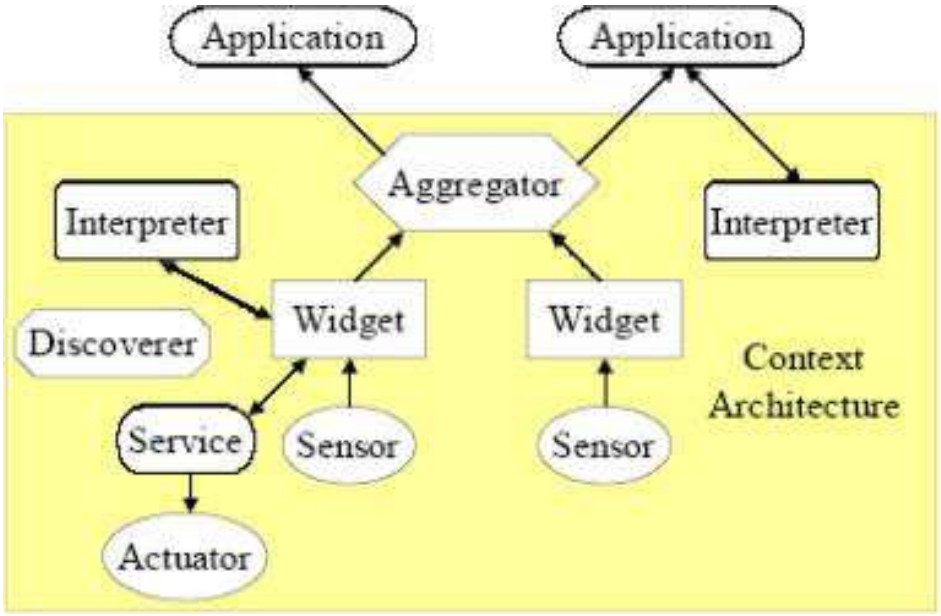


Figure.2.2. Context Toolkit architecture
(Image from (Dey, 2000))

The Context Toolkit has many advantages:

1. Provide a sensor package mechanism.
2. Provide a set of APIs, so that data can be accessed through the network context information.
3. Abstract context data through interpreters.
4. Provide context of data storage including storage of historical information.
5. Provide basic access control to protect privacy of information.

However, some Context Toolkit points and areas need to be improved. Firstly, as an object-oriented modeling approach is used to organize context information, this modeling approach does not provide a universal

context model, not supporting context knowledge sharing and context reasoning. Secondly, Context Toolkit's widgets and aggregators contain their own communication components rather than a middleware communication module to provide communication support for all contexts on the Gateway. Also, the Context interpreter implemented in the Context Toolkit function is relatively simple, just converting one kind of context to another. In fact, the interpreter component should be able to deduce the high-level context. Some inference technology can be added to improve the interpreter component. Finally, the Context Toolkit provides a discoverer for interpreters and aggregators, to find and handle context by application. In reality, as interpreters, and aggregators change dynamically, the Context Toolkit discoverer cannot meet these dynamically changing needs.

2.2.2 CASS

The Context-Awareness Sub-Structure middleware (CASS), developed by Trinity College's Patrick, (Fahy and Clarke, 2004), is a server based context-aware middleware. CASS is designed for context-aware application on hand-held computing devices and other small mobile computing devices, supporting low-level sensors and other context-sensitive input devices while at the same time occupying fewer computing resources.

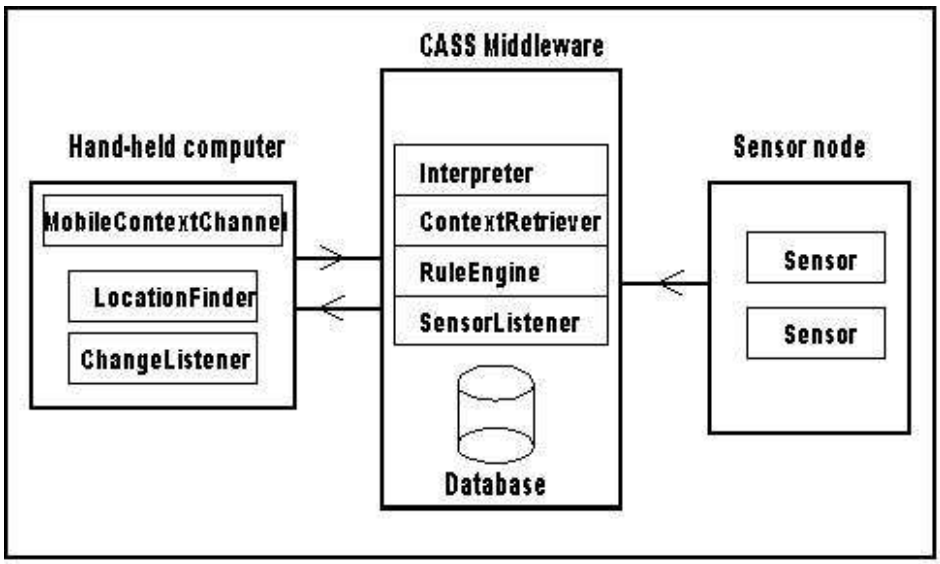


Figure.2.3. CASS architecture
(Image from (Fahy and Clarke, 2004))

CASS middleware contains four main components: Interpreter, Context Retriever, Rule Engine, and Sensor Listener. The rule engine is

used to find a matching goal or goals when a change in context is detected and to infer the high-level context based on the rules stored in the Knowledge base. The Sensor Listener has a communication component, designed to listen for updates from distributed sensor nodes and store context information, while the Context Retriever retrieves stored context. The Context Retriever is responsible for indexing and retrieving the stored context information. The interpreter provides the services for the Sensor Listener and Context Retriever.

CASS has two main contributions: one is supporting high-level context data abstraction. The inference engine adopts the forward chaining technique, a search technique useful for situations in which the search space is wide with many potential goals. The other is separating the knowledge base from the context inference engine. This separation makes it possible to represent knowledge in a more natural fashion, particularly for domain experts without any knowledge of software. The knowledge base can be updated and changed without changing inference engine implementation. However, CASS also has some shortcomings. It does not provide a common context model, as it is not conducive to context knowledge sharing. CASS is not involved in system dynamical extension and context discovery, so it is not easy to develop complex context-aware applications.

2.2.3 CoBrA

Context Broker Architecture (CoBrA), developed by (H. Chen et al., 2004), is an agent-based architecture for supporting context-aware systems in smart spaces. It adopts ontology to model context information, which is suitable for knowledge sharing and high-level context reasoning. CoBrA consists of four functional components: Context Knowledge Base, Context Reasoning Engine, Context Acquisition Module, and Policy Management Module. The architecture diagram of CoBrA is shown below.

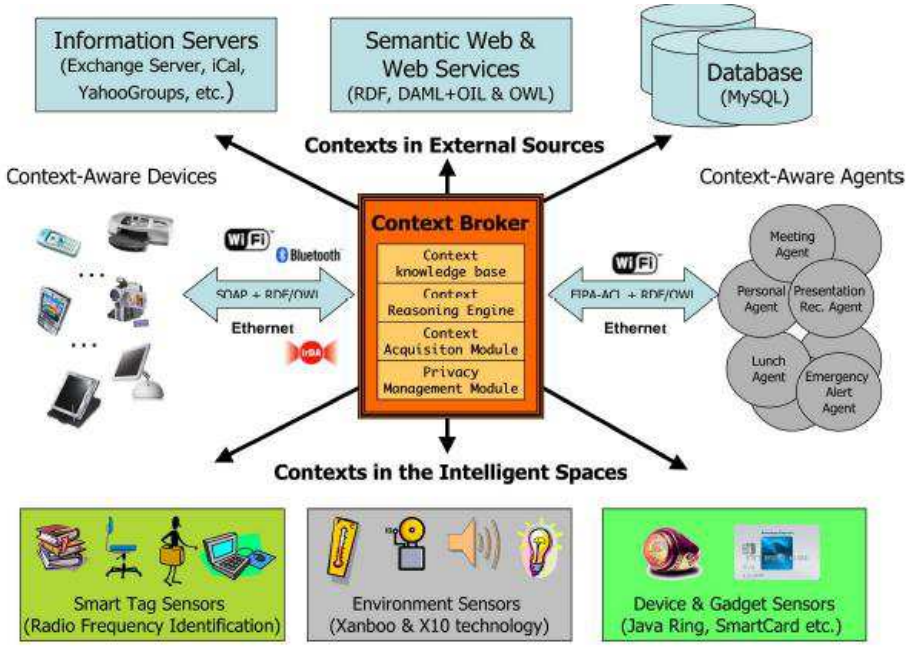


Figure.2.4. CoBrA architecture
(Image from (Chen et al., 2004))

The Context Knowledge Base is a component designed to manage storage of context knowledge. It proposes CoBrA Ontology (Harry Chen et al., 2004) to describe contextual information and share context knowledge, which defines concepts for representing actions, agents, devices, meetings, time and space. Moreover, it provides a set of APIs for other components to access context.

The Context Reasoning Engine is a logic inference engine based on ontologies. The inference engine is used in both types of reasoning. Besides detecting and resolving inconsistent knowledge, it can also infer context knowledge that cannot be easily acquired from the physical sensors.

The Context Acquisition Module is a set of library procedures for context acquisition. The function of this component is similar to Context Toolkit's widget, which hides the low-level context sensing implementations from the high-level functional component.

The Policy Management Module is a component managing the user's privacy policy and controlling the sharing of private information according to the principle of proximity and locality. It extends the Rei language to define policy rule. These rules are used to determine whether other computing entities have permission to share some context information.

CoBrA is one of the earliest systems using semantic web technology to support context-aware ubiquitous computing. This architecture uses OWL language to model context and support context reasoning. The core of

this architecture is an intelligent agent known as the context broker that maintains a shared model of context on the behalf of a community of agents, services, and devices in the space. Moreover, it provides privacy protections for users in the space, including extension of a policy language used to define rules to protect the user's private context information. However, CoBrA's context model is relatively simple, and does not take into account connecting users and activities by temporal and spatial relationships.

2.2.4 SOCAM

Service-Oriented Context-Aware Middleware (SOCAM), developed by (Gu et al., 2005b), is an architecture for building and rapid prototyping of context-aware services. SOCAM is designed for context acquisition, context discovery, context interpretation, and context dissemination based on a set of independent services. The architecture is shown below. It consists of five parts: context provider, context interpreter, context database, context-aware services, and service locating services.

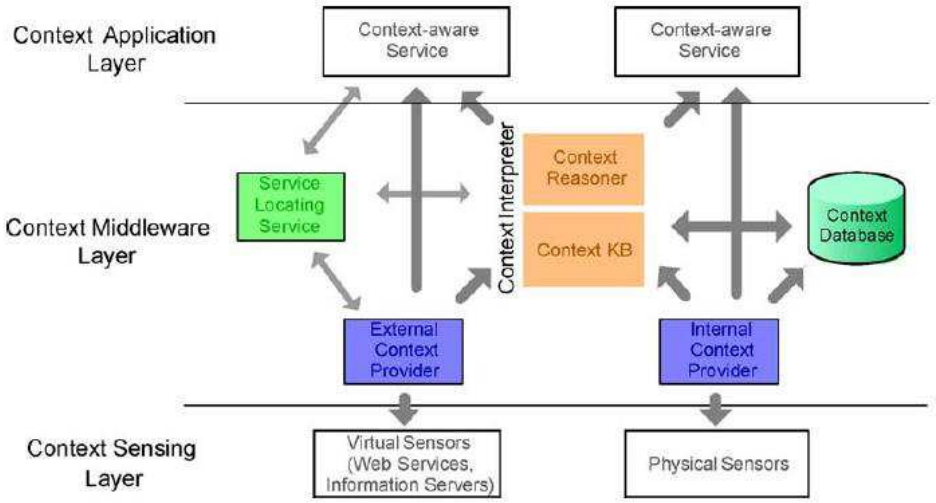


Figure.2.5. SOCAM architecture
(Image from (Gu et al., 2005))

Context providers have two main tasks: they can retrieve contexts from virtual sensors and physical sensors, and can convert raw context to OWL representations to be easily shared and reused by other service components.

The context interpreter is a component that provides logic reasoning services to process context information, which includes inferring high-

level contexts from low-level contexts, querying context knowledge, maintaining consistency of context, and resolving context conflicts. It consists of two parts: a context reasoner and a context knowledge base. The context reasoner supports two kinds of reasoning: ontology reasoning and user-defined rule-based reasoning. Besides containing context ontology and their instances, the context knowledge base provides a set of APIs for other service components to query, add, delete or modify context knowledge.

The context database stores context ontologies and past contexts for a sub-domain.

The context-aware services refer to applications and services making use of different levels of contexts and adapting how they behave according to the current context.

The service locating services propose the service locating service mechanism (Gu et al., 2005a). They can track and adapt to dynamic changes in context provider. They can also enable users or applications to locate these services.

SOCAM provides efficient support for acquiring, discovering, interpreting and accessing various contexts to build context-aware services. SOCAM also uses ontology to model the context similar to CoBrA, but it goes further than CoBrA. It adopts a two-layer hierarchical approach for designing context ontologies, which are divided into the common upper ontology for general concepts and the domain-specific ontology for different sub-domains, Moreover, it adds various characters of context information to context models such as classification and dependency. However, SOCAM merely adopts the rule based on first order logic to determine high-level context, and is inadequate for dealing with complex activity context recognition in Ubiquitous Computing.

2.2.5 CA-SOA

Context-Aware Service-Oriented Architecture (CA-SOA) was developed by (Chen et al., 2006). CA-SOA consists of three components for ubiquitous discovery and access of Web services based on surrounding contexts: an agent platform, a service repository, and the semantic matchmaker. CA-SOA architecture is shown in Figure 2.6.

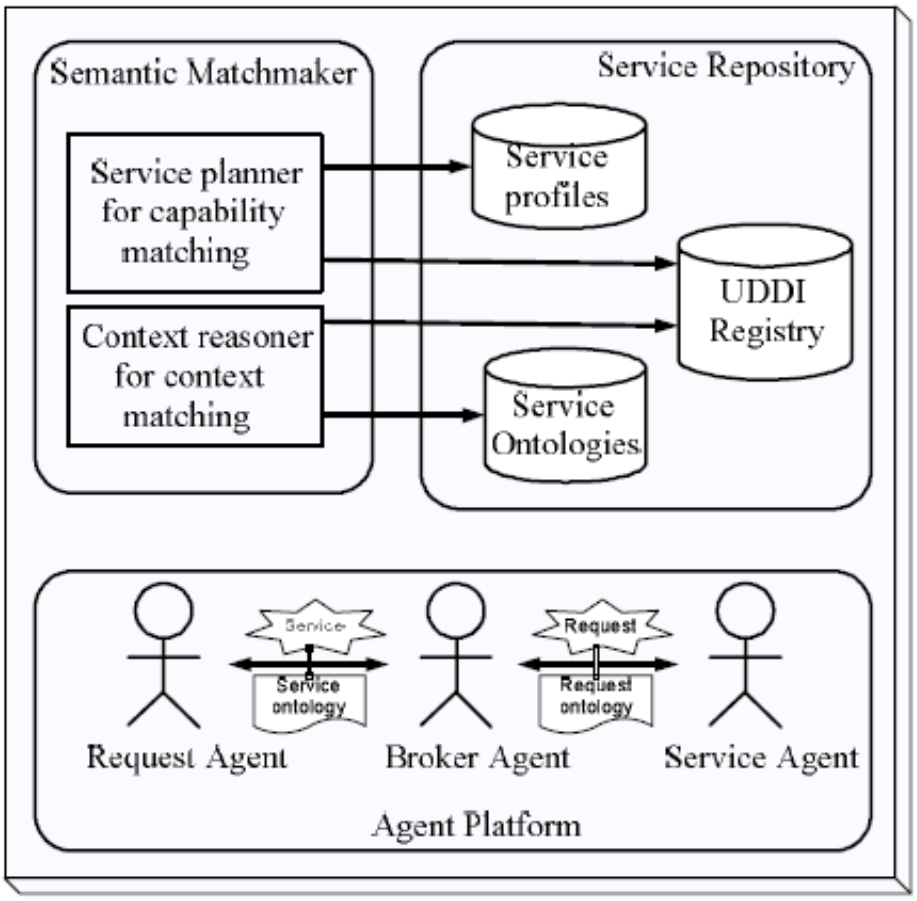


Figure.2.6. CA-SOA architecture
(Image from (Chen et al., 2006))

An agent platform contains three types of agents: service agents, broker agents, and request agents. The agents have been implemented to enhance the context-oriented service description, publication, registration, discovery, and access.

A service repository is designed to encompass a general UDDI Registry associated with service profiles and service ontologies. If the required services are found by the capability matchmaker in the UDDI Registry, the semantic matchmaker will proceed with context matching.

The semantic matchmaker consists of a context reasoner and a service planner. The context reasoner breaks down the service request, based on the requesters' ontology sent along with the service request by the request agent, into a set of sub-requests. The service planner performs a context matching process in order to schedule an integrated composite service based on the broken down request.

Unlike the aforementioned context-aware systems, CA-SOA is developed on the basis of web service. Its main contributions focus on: for-

malizing an ontology-based context model to formally describe and acquire contextual information pertaining to service requesters and services; providing a comprehensive real-time context acquisition method and employing a rule-based matching algorithm with truth maintenance to enhance recall and precision; employing a rule-based matching algorithm with truth maintenance to enhance recall and precision of context-aware service discovery. However, CA-SOA adopts nevertheless a rule-based method to infer high-level context similar to SOCAM, insufficient to deal with complex situations.

2.3 Analysis and Comparison

In this section, we will analyze and compare the aforementioned context-aware system based on five facets: the architecture of the context-aware system, context representation, context sensing, context storage, and context reasoning. The section ends with a conclusion.

In the following section, we will use "+" to denote the kind of characteristic to be supported, "-" to denote the kind of characteristic not to be supported, and "o" to denote the kind of characteristic when it is not clear whether or not it should be supported.

2.3.1 Architecture of the context-aware system

From the general system level viewpoint, we summarize these five architectures of representative context-aware systems based on five characteristics in the table:

- **Context Abstraction:**

The context-aware system hides the complexity of physical sensors or other context resources by providing a high level of abstraction, which enables context resource independence and enhances reusability of architecture components.

- **Communication Model:**

The context-aware system adopts the communication model able to link all the system components working together.

- **Basic Software Component:**

The context-aware system uses the kind of basic software component to achieve various kinds of required functions.

- **Extensibility:**

Whether the device can be added or removed without affecting the entire context-aware system.

- **Reusability:**

Whether the architecture of the context-aware system supports re-using components, simplifying integration and reducing development complexity.

System	Context Abstraction	Communication Model	Basic Software Component	Extensibility	Reusability
Context Toolkit	+	Peer to Peer	Widget	+	+
CASS	+	Client/Server	Object	+	+
CoBrA	+	Client/Server	Agent	+	+
SOCAM	+	Client/Server	o	-	+
CA-SOA	+	Web Service	Service	+	+

Figure.2.7. Analysis and comparison in the context-aware system architecture

2.3.2 Context representations

Context representations pave the way for the context-aware system, including two key facets: context modeling and type of context. An efficient model for handling, sharing and storing contextual information is essential for a working context-aware system. The detailed information for the context model will be addressed in chapter 3. Giving a comprehensive list of context types is a tough task, as it changes according to the specific situation. Location is commonly considered as the most important type of context. We also compare other popular contexts used in context-aware systems, such as calendar, activity, user profile, and device. In addition, the system is required to access historical data in some situations, such as predicting the future context. So we take temporal context into account and se-

lect historical context support as an important characteristic for evaluating these context-aware systems.

System	Context Model	Type of context								Historical Support
		Location	Calendar	Activity	Presence	User Profile	Service	Device	Network	
Context Toolkit	Attribute-value tuple	+	o	+	o	+	o	o	o	+
CASS	Relational data model	+	o	+	o	o	o	o	o	-
CoBrA	Ontology-based model	+	+	+	o	o	o	+	o	-
SOCAM	Ontology-based model	+	-	+	+	+	o	+	-	+
CA-SOA	Ontology-based model	+	+	+	+	+	o	-	-	o

Figure.2.8. Analysis and comparison in context representations

2.3.3 Context Sensing

Contextual information is extracted from context resources, containing three main types: physical sensors, virtual sensors and form filling. Normally, sensors are distributed in a network, where each sensor can send the

collected context data to other components relying on low-level communication protocols. To evaluate the performance of context sensing, we focus on five characteristics shown in the table:

- **Type of Sensing:**

The context-aware system adopts the type of context sensing, which determines the basic performance of context sensing.

- **Mode:**

Two types of mode are concerned: manual and automation, depending on the different types of context resource. In general, manual is used by form filling, while automation is used by physical sensors and virtual sensors.

- **Resource Discovery:**

This is concerned with how to search for and find an appropriate context resource at runtime. This task is performed using different approaches for different context-aware systems.

- **Data Retrieval:**

There are two types of methods for data retrieval: push and pull. When satisfying a condition, the context resource pushes the required context data to the context-aware system, or when the application requires some context, the context-aware system pulls the required context from the context resource.

- **Communication:**

The context-aware system collects context data from the context resource based on communication protocols. The communication protocol varies according to the context-aware system.

System	Type of Sensing	Mode		Resource Discovery	Data Retrieval		Communication
		Manuel	Automation		Push	Pull	
Context Toolkit	Sensor nodes	o	+	-	o	o	TCP/IP
CASS	Sensor nodes	o	+	-	o	+	WiFI
CoBrA	Context acquisition components	o	+	Context-acquisition module	o	o	o
SOCAM	Context providers	o	+	Service Locating Service	+	+	RMI
CA-SOA	Service providers	+	+	UDDI			Web Service

Figure.2.9. Analysis and comparison in context sensing

2.3.4 Context Storage

Storage for context-aware systems does not resemble storage for traditional software systems. Context data contain not only the value but also special knowledge, which eases the application designer's work enormously (Baldauf et al., 2007). The traditional software system normally adopts the relational database to organize and manage data, whereas the context-aware system seldom adopts the relational database. The type of storage database depends on the type of context model adopted by the context-aware system, and the same applies to the request specification.

System	Storage Database	Request Specification
Context Toolkit	Attribute-value tuple	SQL
CASS	Relational	SQL
CoBrA	OWL	SPARQL
SOCAM	OWL	SPARQL
CA-SOA	OWL	SPARQL

Figure.2.10. Analysis and comparison in context storage

2.3.5 Context Reasoning

Context reasoning, also known as context inference, is one of the most important parts of context processing as well as the intellectual component in the context-aware system, designed to automatically further deduce previously implicit facts from explicitly given context information. It focuses on two main tasks: the first is to check and solve inconsistencies in raw data. Due to the widespread adoption of ontology in context-aware systems, it provides an adequate solution for this task. The second task is to infer high-level context information from low-level context data. Rules are always used to achieve this goal, but are inadequate to deal with some complex situations: some specific reasoning methods, such as decision trees, Bayes networks, are involved.

System	Reasoning Capability Support	Ontology-based Reasoning	Specific Reasoning
Context Toolkit	-	-	-
CASS	+	-	-
CoBrA	+	+	Rules
SOCAM	+	+	Rules
CA-SOA	+	+	Rules

Figure.2.11. Analysis and comparison in context reasoning

2.3.6 Discussion

As described above, the context-aware system with respect to architecture is broken down into three critical characteristics: context abstraction, extensibility, and reusability. These five representative context-aware systems have chosen different types of communication models and basic software components, respectively. However, almost all of them support characteristics of context abstraction, extensibility and reusability. Since the ubiquitous computing environment is changing more rapidly than before, supporting extensibility and reusability is even more important. Another trend of the context-aware system is that, without exception, all recent systems separate context resources from context processing, contributing to rapid development of context-aware applications in contrast with earlier context-aware systems. Based on earlier analysis and comparison, we have summarized the key characteristics for context-aware systems, which have become important criteria. The detailed information is listed below:

- **Widespread use of ontology-based models in context representation**

  The context-aware system has to support as many types of context as possible to suit different situations. However, it is not possible to provide all types of context. The ontology-based model provides effective solutions, supporting both knowledge reuse and sharing as well as context inference, so that increasingly more context-aware systems adopt this model to organize contextual information. Furthermore, researchers begin to draw attention to historical context. These context data can be used to predict users' behaviors. Apparently, this will be one of the new directions in the future.

- **Multi-functions**

  Diversity sensors require that the context-aware system supports diverse context sensing technologies. The type of resource discovery depends on the type of communication. Each context-aware system has its own approach, while it is a trend for mode and data retrieval that context-aware systems should support all kinds of mode and retrieval. Consequently, for context-aware systems, a multi-function context sensing component is necessary.

- **Context Storage**

  Unlike other software systems, context-aware systems should store not only data but also relationships. It also can be considered as one kind of knowledge base. As described above, due to its good organizing context features, the ontology-based model has been widely used in context-aware systems recently. So, OWL is mainly used as a storage database, replacing the rational database. Accordingly, the request specification adopts SPARQL rather than SQL.

- **Context reasoning**

  Context reasoning, also known as context inference, plays an increasingly important role in context-aware systems. It is easily observed that three recent context-aware systems in comparison (CoBrA, SOCAM and CA-SOA) provide a support for ontology-based reasoning. This provides a solution for the partial task of context reasoning: check context consistency. Nowadays, the context with which most researchers are concerned has shifted from location to other high-level contexts, such as activity. This requires supporting by context reasoning. However, up to now, there was no practical way to take advantage of this opportunity.

2.4 Conclusion

This chapter reviews the state of art in context-aware computing. After concluding as to the general character of context-aware systems, the five chosen representative context-aware systems are analyzed. We then provide a more detailed analysis and comparison in five fundamental aspects in context-aware systems: architecture, context representations, context sensing, context storage, and context reasoning. This chapter ends with a discussion on the five aspects, indicating the future directions of our work.

3 The Spatial-temporal Ontology-based Model

3.1 Introduction

A context model is needed to define and store context data in a machine processable form (Baldauf et al., 2007). A well designed model is a key accessor to the context in any context-aware system (Strang and Linnhoff-Popien, 2004). The importance of context models for context-aware systems cannot be stressed enough.

In this chapter, firstly the existing context modeling approaches will be surveyed, along with comparison and analysis. Secondly, the detailed description of the ontology-based model is presented. Then, we propose a spatial-temporal context-aware model used to organize the context for different situations. Following this model, the associated technical tools are outlined as well. Finally, this chapter ends with a summary.

3.2 Context modeling

Multiplicity of context leads to many different methods of context representation and different levels of inner links. It is still a challenge to build an abstract logical model based on this complex heterogeneous information, ensuring that context is easily expressed, reasoned and shared.

3.2.1 Context classification

In Ubiquitous Computing, it is necessary to classify context to meet the need for numerous heterogeneous context information. Contextual classification can help context-aware application designers and developers cover the entire possible context and simplify operation of context, whilst contextual classification would help to provide quality context information. In real applications, some types of context are more important than others, such as location, identification, activities and time. These types of context are the main contexts used to characterize the situation of the entity, which not only solve problems of "who" "what" "when" and "where", but also provide an index for other type of context.

Many researchers have attempted to classify context from the different conceptual points of view. Gwizdkal (Gwizdka, 2000) extend the definition of context from Schilit et al. (Schilit and Theimer, 1994), and make a basic distinction between context that is internal or external to the user. Internal context describes user states, which can be made up of work context (e.g. current projects and their status, status of to-do things, project team), personal events (i.e. events experienced by the user. These events are internalized external events), communication context (i.e. state of interpersonal e-mail communication), emotional state of the user. External context describes the state of the environment, which can be made up of location, proximity to other objects (both people and devices), and temporal context. Elena (Elena et al., 2000) classifies context in material and social circumstances. Material context refers to aspects such as the place of use (in an office, at home, in a museum, in the open air), the device (a workstation, a hand−held device) or the available infrastructure (networks, GPS, infrared), while social context is equally important, related to aspects (being alone or not, who the others are, if this exerts pressure) and personal traits (attitudes, preferences, interests).

The above-mentioned types of contextual classification make an effort towards creating a context model, but these classifications are incomplete and not very clear on context information. Taking into account different requirements of context awareness, we take Day and Abowd's definition of context (Dey and Abowd, 1999a) as a starting point and attempt to classify context based on two dimensions: spatial and temporal.

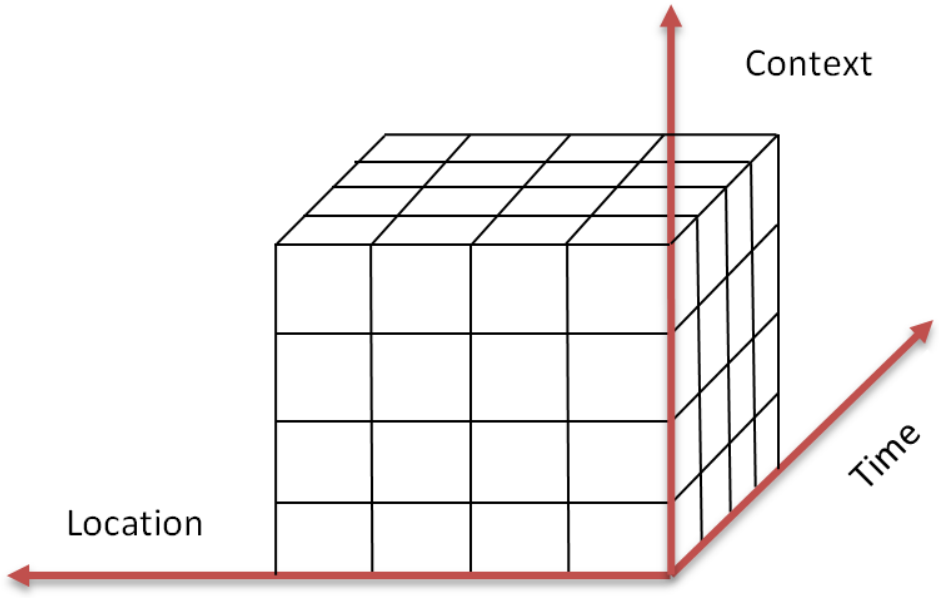


Figure 3.1 The relationship between context, location and time

The spatial context attempts to describe the entity's situation along the space axis. The location context is the central context surrounded by the others. In a different space, location could determine the type of context, for example: in class, user's status comes in two types: student and teacher, while in the office, user's status is totally different.

The temporal context is responsible for storing all contexts in one place during periods or for recording all the contexts on one activity during periods. It provides not only the historical context to context-aware applications, but also a resource to analyze the users' activity, which will be explained in chapter five.

Classification of context on spatial and temporal dimensions clearly defines context in different situations and supports historical context. Meanwhile, it raises a problem, i.e. it is also impossible to provide an exhaustive context list for context-aware designers. For one important entity, the person, he/ she does not stay in one place all the time but moves continuously. To describe his/ her situation requires organizing the entire context according to his/ her changeable locations. A context modeling method is required to model different scenarios and to allow developers to add any new context to this model easily, or even to provide the ability to import and reuse some existing context from another context model. After compar-

ing the existing context modeling approaches, we will provide our own solution.

3.2.2 Existing context modeling approaches

As described earlier, the context model plays a fundamental role in context-aware systems. Due to its collection from heterogeneous context resources, raw contextual information cannot be utilized directly by context-aware applications. Therefore, it requires an approach that can define and organize context in order to be understood by context-aware systems. The context model refers to the approach for describing and organizing contextual data. How efficiently the system uses context largely depends on which type of context model the system adopts. A well designed context model should provide an appreciable interpretation of all kinds of context, easily shared with the required component and conveniently handled by designers.

Currently, there are two main approaches for context modeling: the non-formal and the formal context model. In early research, context-aware systems generally adopt non-formal models, such as Xerox ParcTab (Want et al., 1995) and Olivetti's Active Badge (Want et al., 1992). The Context Toolkit (Salber et al., 1999) uses an object-oriented method to describe context, modeling each type of context as a class. Cooltown (Kindberg et al., 2000) projects made a model for real-world objects such as people, locations and devices using an HTML Web-based method. Each context has an associated Web description. Since the non-formal context model does not support context sharing, reasoning and other functions, it has gradually been replaced by the formal context model in recent years. Entity Relationship (ER) is leveraged to model context. Henricksen et al. (Henricksen et al., 2003) proposed a context model approach combining ER with UML. First-order logic is used to model context in Gaia (Román et al., 2002). CoBra (Chen et al., 2004) and SOCAM (Gu et al., 2005) have adopted ontology to build the context model, and Ranganathan (Ranganathan and Campbell, 2003) designed a middleware for context-aware agents using ontology to define different types of contextual information as well. Due to its good performance on knowledge sharing, context reasoning and context reuse, the ontology-based model is gradually being used extensively.

Strang and Linnhuff-Popien (Strang and Linnhoff-Popien, 2004) have summarized existing context modeling approaches and classified these context models into six types, as follows: Key Value (pairs) Models, Markup Scheme Models, Graphical Models, Object-Oriented Models, Logic-Based Models and Ontology-based Models.

- **Key Value Models**
  The key-value model uses a simple data structure: key-value pairs to represent context, which provides contextual data to applications in the form of variables. The advantage of the key-value model is that it is relatively simple and allows easy data management. However, it is inadequate for processing complex context in sophisticated environments and lacks supporting contextual information retrieval algorithms. The key-value model is used in early context-aware systems, such as (Schilit et al., 1994).

- **Markup Scheme Models**
  Markup scheme models adopt an hierarchical data structure to express contextual information using markup language, such as XML or RDF. Compared with the key-value model, the markup scheme model provides a better context representation, but does not describe relationships between contextual information, and lacks an effective resolution mechanism for processing incomplete, dynamic contextual data.

- **Graphical Models**
  Graphical models are commonly used graph-oriented modeling tools such as UML to describe contextual information. Henricksen et al (Henricksen et al., 2003) have adopted the graphical model to organize context. This model is intuitive and has powerful presentation skills for relational databases in information management systems.

- **Object-Oriented Models**
  Object-oriented models leverage object-oriented technologies: encapsulation and reusability to express contextual information. The context details are packaged in objects. The other components must be accessed through a specific interface to handle context information. A typical project is TEA (Schmidt and Laerhoven, 2001).

- **Logic-Based Models**
  Logic-based models express context as a set of facts, expressions and rules. These models were used for the first time to formalize contextual information in early 1993 by McCarthy and his group (McCarthy, 1993). In general, logic-based models adopt a higher formal representation, but are limited to reusability and applicability.

- **Ontology-based Models**
  Ontology-based models use ontology to represent conceptions of contextual information and relationships between context information. They provide a

unified terminology and concepts, conducive to knowledge sharing. They also adopt the formal representation of knowledge domains, supporting the reasoning mechanism. In addition, they can import the other existing ontology databases for easy reuse of knowledge.

Thomas Strang and Claudia Linnhoff-Popien (Strang and Linnhoff-Popien, 2004) then compared the six types of context modeling on six main requirements in ubiquitous computing systems:

1. **Distributed composition (dc):**
   Any ubiquitous computing system is a derivative of a distributed computing system which lacks a central instance responsible for creation, deployment and maintenance of data and services, and in particular context descriptions. Instead, composition and administration of a context model and its data vary with notably high dynamics in terms of time, network topology and source.

2. **Partial validation (pv):**
   It is highly desirable to be able to partially validate contextual knowledge on a structure as well as on an instance level against a context model in use even if there is no single place or point in time where the contextual knowledge is available on one node as a result of distributed composition. This is particularly important due to the complexity of contextual interrelationships, which make any modeling intention error-prone.

3. **Quality of information and richness (qua):**
   Quality of information delivered by a sensor varies over time. Also, the richness of information provided by different kinds of sensors, characterizing an entity in a ubiquitous computing environment, may differ. Thus, a context model appropriate for usage in ubiquitous computing should inherently support quality and richness indications.

4. **Incompleteness and ambiguity (inc):**
   The set of contextual information available at any point in time, characterizing relevant entities in ubiquitous computing environments, is usually incomplete and/or ambiguous, in particular if this information is gathered from sensor networks. This should be covered by the model, for instance by interpolation of incomplete data on the instance level.

5. **Level of formality (for):**
   It is always a challenge to describe contextual facts and interrelationships in a precise and traceable manner. For instance, to perform the task “print

document on printer near to me", you need to have a precise definition of the terms used in the task, for instance what "near" means to "me". It is highly desirable that each participating party in a ubiquitous computing interaction should share the same interpretation of the data exchanged and the meaning "behind" it (so- called "shared understanding").

6. Applicability to existing environments (app):

From the implementation viewpoint, it is important that a context model is applicable within the existing infrastructure of ubiquitous computing environments, e.g. a service framework such as Web Services.

They use "++" "+" "-" "—" to denote the level of support for a characteristic, "++" meaning the highest supported and "—" meaning the lowest supported. The results are shown in the table below. The ontology-based model shows better performances in the six requirements than the other five context modeling approaches, fulfilling all the requirements and fully supporting distributed composition (dv), partial validation (pv) and level of formality (for).

Approach - Requirem.	dc	pv	qua	inc	for	app
Key-Value Models	-	-	—	—	—	+
Markup Scheme Mod.	+	++	-	-	+	++
Graphical Models	—	-	+	-	+	+
Object Oriented Mod.	++	+	+	+	+	+
Logic Based Models	++	-	-	-	++	—
Ontology Based Mod.	++	++	+	+	++	+

Figure 3.2 Context Modeling Approach preference
(Image from (Strang and Linnhoff-Popien, 2004))

Continuing their work, we study the twenty existing popular context-aware systems over the past two decades: CASS (Fahy and Clarke, 2004), CoBra (Chen et al., 2004), Context Management Framework (Floréen et al., 2005), Context Toolkit (Dey and Abowd, 1999b), CORTEX (Sørensen et al., 2004), Gaia (Román et al., 2002), Hydrogen (Hofer et al., 2003), SOCAM (Gu et al., 2005), CMF (Van Kranenburg et al., 2006), STU21 (Singh and Conway, 2006), Akogrimo(Akogrimo Salerno, 2008), Anyserver (Han et al., 2005), CA-SOA (Chen et al., 2006), ContextUML (Sheng and Benatallah, 2005), CoWSAMI (Athanasopoulos et al., 2008), ESCAPE (Truong et al., 2007), Keidl&Kemper' context framework (Keidl and Kemper, 2004), inContext (Melchiorre et al., 2008), Omnipresent (de

Almeida et al., 2006) and SiWS (Matsumura et al., 2006). According to analysis of their context model, half of them adopt ontology technology to organize contextual data. Especially in recent years, the ontology-based model is extensively used in context-aware systems.

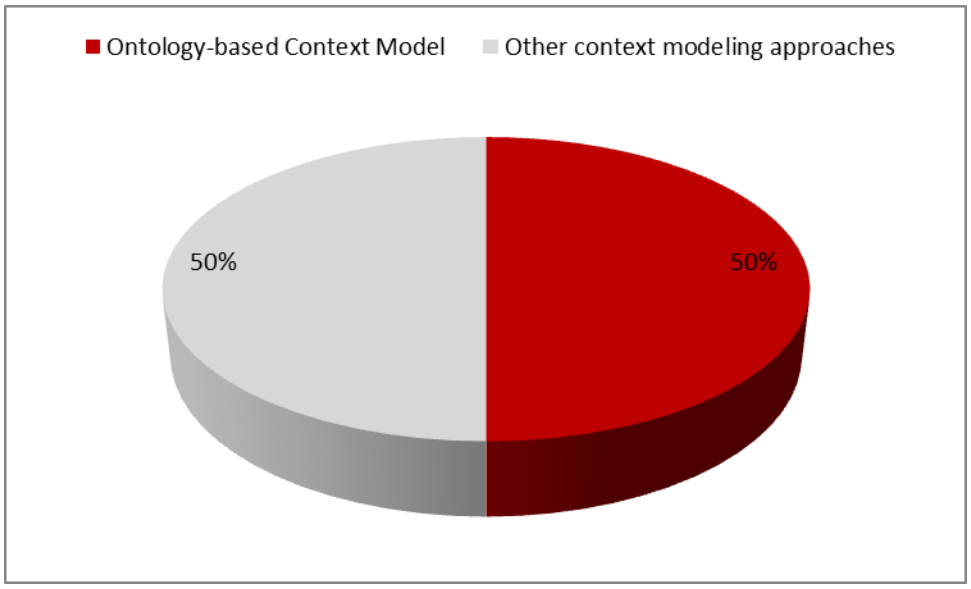


Figure 3.3 Comparisons in Context Modeling Approaches

As analyzed above, ontology is a promising instrument for representing a description of concepts and relationships, and is particularly suitable for project parts of the information describing and being used in our daily life on a data structure utilizable by computers. It provides high and formal expressiveness and the possibilities for applying ontology reasoning techniques. The ontology-based context model is a compelling approach for modeling contextual information. Therefore, our context-aware middleware uses ontologies as an underlying context model technology.

3.3 The Spatial-temporal Ontology-based Model

The context model is one of the most important components in the context-aware system. Diversity of context resources leads to different types of context expressed in their own ways with internal interrelationships. How to establish a unified abstract logical model for this complex heterogeneous information, making this information easily expressed, reasoned and shared, is a very important issue in the context-aware system. According to our earlier analysis, we present a context model based on ontology for both spatial and temporal aspects. This model adopts an hierarchical structure to meet

the different requirements of ubiquitous computing. Besides characterizing the entity's current situation, it stores the historical context data, which can be used to analyze users' activities and provide predictions and recommendations. The ontology-based modeling approach is adopted to organize our context model. It enables a context of information sharing in a specific area by defining a common vocabulary, as well as providing basic concepts for the field and the relationship between these concepts. Context expressed in ontology can be used by users, devices and services. In addition, the ontology-based model allows reuse of domain knowledge, helping a large ontology model to be built by several small ones. We will define ontology and the ontology description language below.

3.3.1 Ontology

Ontology stems from an old philosophical concept, meaning an objective existing system from the philosophy category, and concerns the abstract nature of objective reality (Fu et al., 2009). Over the past decade, research into ontology has increasingly matured and gone beyond the scope of philosophy, concerning many fields, such as artificial intelligence and information technology.

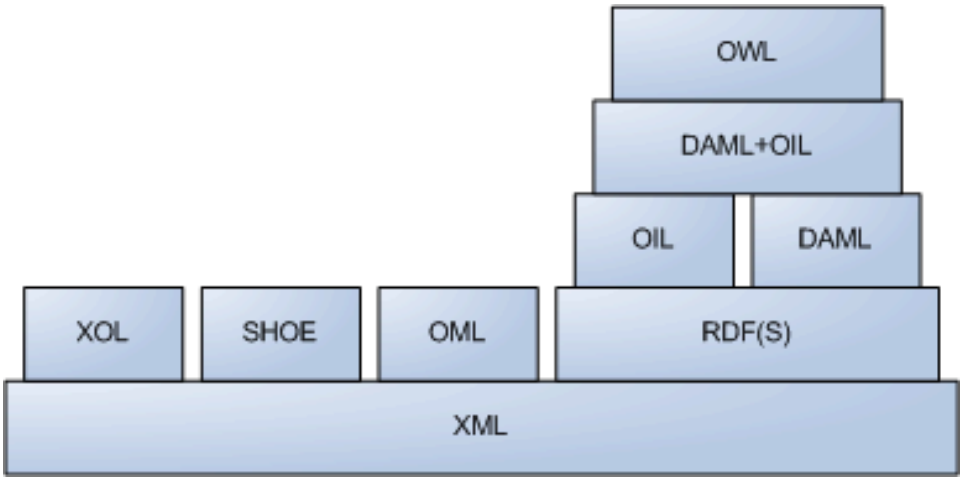
In 1991, Neches, Fikes, et al. (Neches et al., 1991) defined ontology for the first time in the domain of artificial intelligence: "An ontology defines the basic terms and relations comprising the vocabulary of a topic area as well as the rules for combining terms and relations to define extension to vocabulary." According to this definition, ontology states that terms are not only defined explicitly but also derived from rules. A popular definition of ontology is given by Gruber (Gruber, 1993): "ontology is the clear specification on a conceptual model", involving two meanings: first, it refers to abstraction, induction, conceptualized in a field; second, it provides a conceptualization of the results expressed in unified form which can be understood by humans and computers. Based on this definition, Borst (Borst, 1997) took it a step further:" ontology is the clear specification on a sharing conceptual model." In 1998, Studer et al. (Studer et al., 1998) proposed a more detailed alternative, including four aspects:

- **Conceptualization:** models obtained by abstracting some related phenomena in the objective world, which take place independently from the specific state of the environment.
- **Explicit:** explicit definitions for concepts used and their restriction
- **Formal:** the ontology model can be processed by computer
- **Share:** ontology aims at group consensus instead of individual consensus.

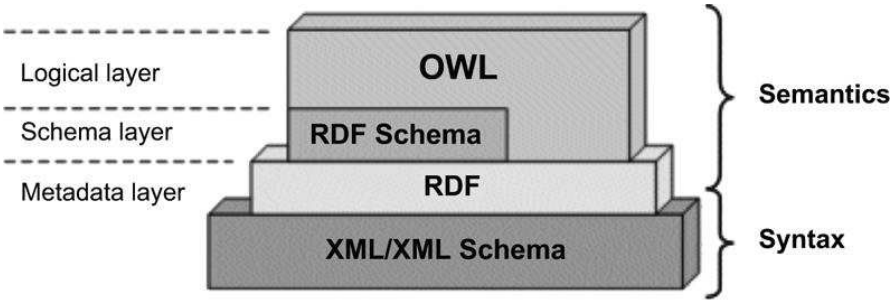
From the above definition of ontology, we find that "ontology" is involved: terminology (Glossary), relationships, rules, formal specification, domain knowledge, expression and sharing. In fact, ontology lays down the basic knowledge structure for a domain by the concepts, terminology, and standardized description of relationships.

3.3.2 Ontology description language

Recently, industry and research institutions have developed a variety of ontology description languages based on different representation forms, shown in Figure 3.4(a). The description language is based on XML (Extensible Markup Language) syntax including: RDFS ("Resource Description Framework," 2004), DAML + OIL (DARPA Agent Markup Language + Ontology Interchange Language) ("DAML+OIL," 2001), OWL ("OWL Web Ontology Language," 2004), etc. The relationship between these languages is shown in Figure 3.4(b). W3C (World Wide Web Consortium) has recommended three of these languages as standard models related to ontology: XML, RDF/RDFS and OWL.



(a) Ontology language family



(b) Relationship between main ontology languages

Figure 3.4 Ontology description languages

- **XML**

  XML is a markup language that defines a set of rules for encoding documents in a unified format ("XML," 2013). It is a textual data format with strong support via Unicode for the world's languages, emphasizing simplicity, generality, and usability over the Internet. XML technology is a collection of standards and protocols recommended and approved by W3C, ranging from the underlying network to the application layer involving data representation, data processing, message exchanging and delivering etc. It contains several other related specifications, gratis open standards, such as underlying standards: DTD (Document Type Definitions), style standards: CSS (Cascading Style Sheets), query criteria: XQL (XML Query Language), and parsing standard: DOM (Document Object Model). XML paves the way for other ontology description languages, providing a well-structured base.

- **RDF/RDF Schema**

  RDF (Resource Description Framework) locates above on the XML in the ontology language stack. RDF is a language for representing information about resources in the World Wide Web, containing Resources, Properties and Statements. Statements consist of three components: Subject, Predicate and Object. RDF provides a data model to express web metadata, using objects and their relationships. RDF defines not only a triple pattern, consisting of object, attribute and value, as basic modeling primitives, but also a standard syntax representing a triple pattern. According to a characteristic of RDF, any complex relationship can be expressed as a composition of several simple dual-relationships. Therefore, the RDF data model can be

used to describe all the resources on the web. RDF provides a human-readable and machine-readable format to exchange between system and application, overcoming XML's shortcoming of lack of semantics.

RDFS (RDF Schema) completes RDFS. It adds some new semantic primitives based on XML, extending the data semantic description. RDFS provides a simple machine-readable semantic model and forms the basis for other ontology description languages. However, it does not take semantic conflict into account.

- **DAML-OIL**

XML and RDF can provide a simple solution for ontology description but have only a limited ability in expression. Therefore, researchers have sought to develop a more expressive model language. OIL (Ontology Interchange Language) is one of them, developed by the on-To-Knowledge program in Europe. This model language extends syntax and description logic of RDF and joint description logic, framework language and web standards all together for the first time.

The DAML (DARPA Agent Markup Language) was the name of a US funding program at the US Defense Advanced Research Projects Agency (DARPA) started in 1999. Many United States research institutions were involved in this study. It is an extension of RDF based on object-oriented technology. DAML's earliest version was called DAML-ONT, before it changed to DAML+OIL. It is a language following on from DAML and OIL that combines the best features of both. DAML + OIL describe the domain structure by classes and attributes, enhancing the ability of language description. Unlike RDF, DAML+OIL is not a data model but a structure language for describing the RDF data model. To some extent it can be considered as another RDF language, so it inherits RDF's weakness as well.

- **OWL**

OWL (Web Ontology Language) is a family of knowledge representation languages for authoring ontologies. It has been endorsed by the director of W3C as a W3C Recommendation ("OWL Web Ontology Language," 2004), located at the top of the ontology language stack. OWL started as an improvement of DAML-OIL, drawing on its design and application experience to enrich the semantic definition mechanism. OWL, in the abstract syntax, contains a sequence of annotations, axioms, and facts. Annotations on OWL ontologies can be used to record authorship and other information associated with ontology, including import references to other ontologies.

The main content of OWL ontology is conveyed in its axioms and facts, which provide information about classes, properties, and individuals in the ontology ("OWL Web Ontology Language," 2004).

OWL provides more means of expression than XML, and RDF/RDFS in semantic expression, adding more vocabulary for describing properties and classes: among others, relationships between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties and characteristics of properties (e.g. symmetry), and enumerated classes. Therefore, OWL provides a strong expressive power of ontology.

Focusing on different requirements, OWL provides three increasingly expressive sublanguages: OWL Lite, OWL DL, and OWL Full.

OWL Lite: provides the primary and basic functional subset of OWL for users. It supports simple constraint functions and is relatively easy to imply. W3C uses a simple example to interpret this: OWL Lite supports cardinality, but only permits cardinality values of 0 or 1. There are more development tools supporting OWL lite than the other two sublanguages. Developing the process is relatively easy.

OWL DL: an extension of OWL Lite, more expressive than OWL Lite. OWL DL contains all OWL language constructs, but they can be used only under certain restrictions. W3C uses a simple example to interpret this: a class may be a subclass of many classes, a class cannot be an instance of another class. OWL DL provides a description logic reasoning for applications requiring maximum expressiveness while retaining computational completeness and decidability, but its vocabulary is limited.

OWL Full: possesses the most expressive OWL sublanguage, considered as an extension of OWL DL. It is designed for users requiring maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. W3C uses a simple example to interpret this: a class can be treated simultaneously as a collection of individuals and as an individual in its own right. It allows new ontology to be added to predefined RDF, OWL RDFS vocabulary. Therefore, no reasoning software could support complete reasoning for every feature of OWL Full.

As analyzed above, OWL has been chosen to construct our spatial - temporal context model. As the latest knowledge representation language endorsed by W3C, it derives advantages from previous ontology description languages, building on XML's ability to define customized tagging schemes and RDF's flexible approach to representing data. OWL provides a common understanding of contextual information to facilitate context modeling

and reasoning on imperfect and ambiguous contextual information and to enable context knowledge sharing and reuse.

3.3.3 Spatial-temporal context model

The spatial and temporal context model describes the entity's situation in space and time. Context-aware research has taken account of spatial dependence among different scenarios: meeting room, living room, office. However, context is not only connected in the spatial aspect, but also dependent in the form of a time series. At all times context-aware systems obtain contextual information from the heterogeneous context resource, which is a description of the situation of entities at any one time. Context-aware applications always require a description of the situation of entities at all times. In other words, the context model must take account of temporal correlations as well as special correlations. Both the spatial and temporal models are necessary when context is collected across space as well as time.

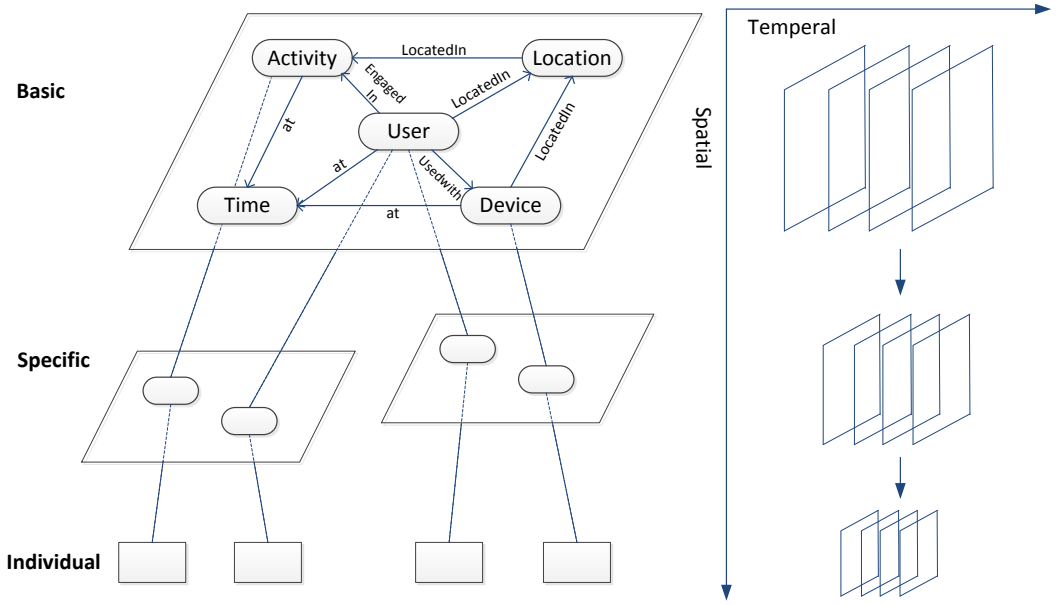


Figure 3.5 The Spatial-temporal Ontology-based Model

Spatial context refers to a geographical area where data were collected, a place which is the subject of a collection, or a location which is the focus of an activity. Spatial context can be either a point location or an area for a specific scenario, such as a smart room, office room or hospital. From the spatial dimension, we employ an hierarchical structure to describe the us-

er's situation and circumstance based on OWL. The structure is shown in Figure 3.5. In this context model, the ontology context is broken down into two layers: the basic model and the specific model. The basic model defines generic conceptions and relationships in AmI, which comes up with a basic context structure. It has five interrelated basic classes: user, location, time, activity, and device, and eight properties (relationships) between classes are identified, which represent who, where, when, what, and how:

- Who: the user communicating with the context-aware system.
- Where: refers to location information that is a crucial context in context-aware computing.
- What: the context-aware system needs to know what activity users are involved in.
- When: refers to time, which can be used as an index to capture information: we will discuss this more thoroughly in the next section.
- How: how the system can acquire this context, in other words, which devices are involved in this situation.

In ubiquitous computing environments, context-aware applications or services are typically used for different groups of sub-fields of intelligent environment collections, such as residential domain, office domain and transport domain. Each domain uses the same basic ontology for the common concept, but they have a huge difference in detail. General context-aware ontology can be completed and upgraded by more precise information related to a particular application or application area, known as specific ontology. This is used to define the details and attributes of the sub-domain, which can be reused by different applications. The new specific ontology will be added to the basic model to construct the new context model when the environment changes. To give an example: when a user leaves his/ her residence for driving, the transport domain ontology needs to be added to the context model. Figure 3.6 shows the relationships between basic ontology, specific ontology, and applications. Easily observed, the specific ontologies can be shared in different context-aware applications, while a context-aware application can use multiple specific ontologies to build the context model. To conclude, this hierarchical structure enables flexible and extensible context models.

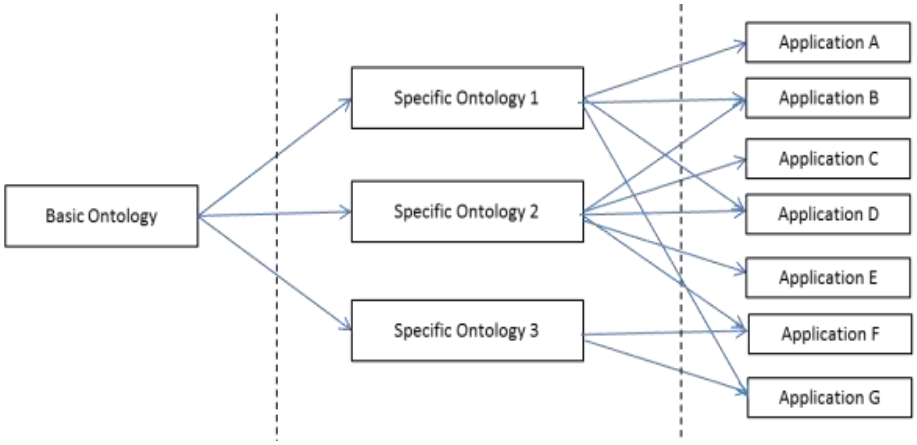


Figure 3.6 Relationship between context model and application

This context model uses OWL to provide a description of each physical or conceptual object, including users, activities, computing device location information, as well as virtual entities: users' profiles. The "class" in OWL is used to represent a type of entity. The relationship between the entities and their attributes can be described as "owl: DatatypeProperty", while the relationship between entities can be represented as "owl: Objectproperty". The OWL property (owl: subClassof) is used to indicate the subclass entity in the hierarchical structure, allowing a new specific ontology to be added to basic ontology as an extension. A simple example is shown in Figure 3.7. "Student" is a sub-class of the class: "User", and is disjointed with the class "Teacher" in Figure 3.7(a). The individual "Tao" with student ID "UL0601" is an instance of the class "Student" shown in Figure 3.7(b). Figures 3.7 (c) and (d) show how Datatype properties and Object properties are represented. Restrictions can also be defined to constrain the Classes.

```
<owl: Class rdf:ID = "Student">
     <rdfs:subClassOf>
        <owl:Class rdf:ID = "User">
     </rdfs:subClassOf>
     <owl:disjionWith>
        <owl: class rdf:ID ="Teacher">
     </owl:disjionWith>
</owl:Class>
```

(a) Class

```
<Student rdf:about = "#">
      <hasStudentID
      rdf:datatype = " http://www.w3.org/2001/XML
      Schema#string"> UL0601 </hasStudentID>
      <hasName
      rdf:datatype = "http://www.w3.org/2001/XML
      Schema#string"> Tao </hasName>
</Student>
```

(b) Individual (instance)

```
<owl:DatatypeProperty
      rdf:about = "#hasName">
      <rdfs:domain
      rdf:resource = "User"/>
      <rdfs:range
      rdf:resource = "http://www.w3.org/2001
      /XMLSchema#string">
</owl:DatatypeProperty>
```

(c) Datatype Property

```
<owl:ObjectProperty
      rdf:about = "#useWith>
      <rdfs:domain rdf:resource = "User"/>
      <rdfs:range rdf:resource = "Device">
      <owl:inverseOf>
         <owl:ObjectProperty
         rdf:about = "#isUsedBy" />
      </owl:inverseOf>
</owl:ObjectProperty>
```

(d) Object Property

Figure 3.7 Class, Individual and Property in OWL format

In the ubiquitous computing environment, each object is viewed as an entity in the context model, for example: a person, meeting room and campus. Meanwhile, there are some physical relationships between them. Therefore, context is represented as a triple pattern: {*entity (subject), predicate, entity (object)*} with OWL format in the context model. This is motivated by human language, used as an analogy of the English sentence. This can be simply explained: {Tao, Locatedin, Classroom 3}, "Tao" is an entity, belonging to Class: Person; "Classroom 3" is an entity, belonging to Class: Location, and "Locatedin" is a predicate, connecting the entity "Tao" and the entity "Classroom 3". "Predicate" is a property representing a directed binary relationship that specifies class characteristics. It may possess logical capabilities such as being transitive, symmetric, inverse and functional. We take the situation "Tao LocatedIn Classroom 3" as an example again. The "LocatedIn" is a predicate, with the logical feature: transitive. The sys-

tem can know “Tao Located in Building 2”, because “Classroom 3” is in “Building 2” and “LocatedIn” is transitive. Another example: the context is “Tao TalkTo Yun”. The system can know “Yun TalkTo Tao” as well, because the predicate “TalkTo” has the logical capability: symmetric. By means of the triple pattern, a complex smart space can be modeled with ample simple connected contextual information.

The context temporal model is an attempt to organize context according to a temporal dimension, including a time period during which context was collected, or a time period that an activity or a collection is linked to intellectually or thematically. It relates to two primary classes: time and device. The time class provides three subclasses to describe the temporal relationship between user and activity. The device class uses the subclass: the historical sensed to record the historical context data, which can be used to analyze users’ activities and provide predictions and recommendations.

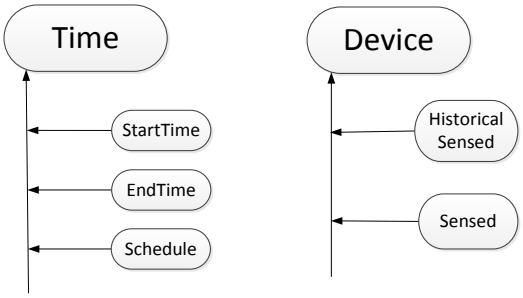


Figure 3.8 partial classes in temporary model

Besides storing the historical context for context-aware application, but also provides a context extracting mechanism based on temporal view.

Depend on different sources, the context is divided into three categories: 1. pre-defined context.2 detected context: 3.inferred context. Predefined context refer to context that is put in directly by end-users or application designers, such as user’s profile context, specific environment context; Detected context is get from sensors or web, as well namely low-level context; inferred context is deduced by inference engine, considered as high-level context.

We design a context extracting mechanism based on temporal view, including three approaches: 1.required context; 2.high-frequency context; 3.long-term context. Required context refers that the context is pulled from context resource when context-aware system requiring, mainly

used for inferred context and some detected context not changing frequently, like: temperature. High-frequency context is pushed to context-aware system when the context- is changed, mainly used for detected context, such as: location. Long-term context refers that the context is relative static, updated by user or application, mainly used for predefine context, like: user profile.

Finally, We use the simple Figure 3.9 to explain the relationship between the different developer and ontology model. The basic model is defined by Middleware developer. The application developer can propose specific model to extend basic model for different and complex situation. Application instances are yielded by end user, which are with basic model and specific model to describe the user's situation in the ubiquitous environments.

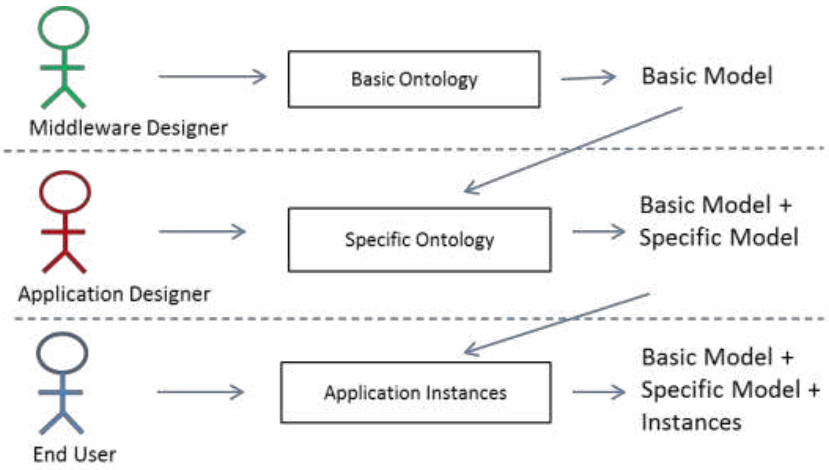


Figure 3.9 Different contributors in context modeling

3.3.4 Ontology-based model development tools

As described above, we have chosen an ontology description language, OWL, to construct the spatial-temporal context model. Though OWL is an XML-based language, development of OWL is different from a conventional XML. Facing heterogeneous sensor environment, we employ the Protégé Ontology editor and Jena to develop OWL in Java. Generally, the Protégé Ontology editor provides a feasible platform supporting WYSIWYG to construct the initial context model. Jena provides a set of APIs that enable developers to handle the context in JAVA. Detailed information will be provided below.

3.3.4.1 The Protégé editor

The Protégé editor, developed by the Stanford Center for Biomedical Informatics Research, is a free, open source ontology editor and knowledge-based framework (framework "Protégé," 2004). The Protégé platform supports modeling ontologies via a web client or a desktop client, which is based on Java, is extensible, and provides a plug-and-play environment making it a flexible base for rapid prototyping and application development.

The Protégé platform supports two main modeling ontology methods: the Protégé-Frames editor and the Protégé-OWL editor.

The Protégé-Frames editor enables users to build and handle ontologies that are frame-based, in accordance with the Open Knowledge Base Connectivity protocol (OKBC). In this model, ontology consists of a set of classes organized in a subsumption hierarchy to represent a domain's salient concepts, a set of slots associated with classes to describe their properties and relationships, and a set of instances of those classes - individual exemplars of the concepts that hold specific values for their properties.

The Protégé-OWL editor enables users to build ontologies for the Semantic Web. It implements a rich set of knowledge-modeling structures and actions that support the creation, visualization, and handling of ontologies in various representation formats including OWL, RDF(S), and XML Schema. It can be customized to provide domain-friendly support for creating knowledge models and entering data. The Protégé-OWL editor enables users to:

- Load and save OWL and RDF ontologies.
- Edit and visualize classes, properties, and SWRL rules.
- Define logical class characteristics such as OWL expressions.
- Execute reasoners such as description logic classifiers.
- Edit OWL individuals for Semantic Web markup.

Protégé-OWL's flexible architecture makes it easy to configure and extend the tool. Protégé-OWL is tightly integrated with Jena and has an open-source Java API for the development of custom-made user interface components or arbitrary Semantic Web services. In addition, Protégé is supported by a strong community of developers and academic, government and corporate users, who use Protégé for knowledge solutions in areas as diverse as biomedicine, intelligence gathering, and corporate modeling. In our system, it is used to construct our initial context model.

3.3.4.2 Jena

Jena, supported by the Apache Software Foundation, is a Java framework for building Semantic Web applications providing a collection of tools and Java libraries to help develop semantic web and linked-data apps, tools and servers ("Apache Jena," 2010) .

Via the Ontology API, Jena aims to provide a consistent programming interface for ontology application development, independently from which ontology language is used in programs.

Jena can be used to create and handle RDF graphs. Jena has object classes to represent graphs, resources, properties and literals. The interfaces representing resources, properties and literals are called Resource, Property and Literal respectively. In Jena, a graph is called a model and is represented by the model interface. An ontology model is an extension of the Jena RDF model, providing extra capabilities for handling ontologies. Ontology models are created via the Jena ModelFactory. The fundamental concept in ontology is class, property, and individual. Jena provides a feasible solution to handle these in Java.

Classes are the basic building blocks of ontology. A simple class is represented in Jena by an OntClass object. An ontology class is a facet of an RDF resource. One way, therefore, to obtain an ontology class is to convert a plain RDF resource into its class facet.

Property denotes the name of a relationship between resources, or between a resource and a data value in ontology. It corresponds to a predicate in logic representations. Ontology applications need to store, retrieve and make assertions about properties directly. So, Jena has a set of Java classes that allow users to conveniently handle the properties represented in an ontology model. A property in an ontology model is an extension of the core Jena API class Property and allows access to the additional information that can be asserted about properties in an ontology language. The common API super-class for representing ontology properties in Java is OntProperty.

In OWL Full, any value can be an individual – and thus the subject of triples in the RDF graph are ontology declarations. In OWL Lite and DL, the language terms and the instance data that the application is working with are kept separate, by definition of the language. Jena therefore supports a simple notion of an Individual, which is essentially an alias for a Resource. While Individuals are largely synonymous with Resources, they provide a programming interface that is consistent with the other Java classes in the ontology API.

Furthermore, Jena supports another good feature: ontology inference, the details of which will be given in chapter five on intelligent context inference.

3.4 Conclusion

In this chapter, the study focuses on the context model. This model organizes context in machine readable format, laying the groundwork for the context-aware system. After reclassifying context and analyzing the existing context model approach, we propose a spatial-temporal ontology-based model. This model provides a solution to organize context characterizing the entity's situation along a temporal and spatial axis. Developers can easily use this model to define specific scenarios and obtain historical support. The model adopts an hierarchical structure, thus enabling different developers to create and modify different levels of model. The ontology technique allows support of context share, context reuse, and context inference.

4 Context-aware Middleware in Ambient Intelligence

4.1 Introduction

Ambient intelligence (AmI) refers to a numerically augmented real environment that is sensitive and responsive to the presence of people ("Ambient intelligence," 2013). In an ambient intelligence world, communication objects and devices work together to support people in carrying out their everyday (or professional) life activities and tasks in an easy, natural way using information and intelligence hidden in the network of connecting objects and devices. AmI corresponds to a new vision of daily (or professional) life, consisting of different kinds of sensors, actuators, communication objects and computing devices, which lead to pervasive intelligence in the surrounding environment supporting users' activities and interactions. This means that computing technology will exist in everything that surrounds us (devices, appliances, objects, clothes, materials), and that everything will be interconnected by a ubiquitous network. The system formed by all these interconnected intelligent things (also called the Internet of Things) interacts with humans by means of advanced interfaces, which are natural, flexible and adaptable to the needs and preferences of each user. The final goal is to acquire an adaptive and "intelligent" system that assists humans in their daily (or professional) activities.

To attain AmI, first you need to integrate all sensors, actuators, communication objects and computing devices to the system. Low level mechanisms and drivers are necessary, but must be encapsulated in a more common and higher level view making it possible to collect, process and

propagate information between these components in an appropriate manner, i.e. take into account changes in context, and propagate appropriate decisions. In computing, this kind of concern is known as "context-aware computing" (Kjær, 2007), which was first discussed by Schilit and Theimer in 1994 (Schilit and Theimer, 1994). This kind of system that is related to its physical environment needs to adapt its behavior according to context management and adaptation to deal with the different resources present in the environments (Baldauf et al., 2007). Overall middleware characteristics (multiple sources and multiple destination information management) must be adapted to context-aware computing in order to create an appropriate and efficient environment. To find a solution to develop context-aware application, we propose a context-aware middleware to support AmI. Unlike traditional middleware, our context-aware middleware is designed in a two-layer structure. This consists of a low layer based on a sort of ESB (Enterprise Service Bus), in charge of communicating in standard manner with in-environment sensors and actuators and of managing their dynamic discovery. Then a high layer, known as the VCI (Versatile Context Interpreter), provides higher and semantic data interpretations. Our context-aware middleware ensures rapid development of new context-aware applications.

In the remainder of this chapter, an AmI situation scenario is presented, allowing contextualization following a technical explanation. Then, several aspects of our middleware are explained, starting with a low layer satisfying context collecting requirements. The next step is a description of a high layer, focusing on context processing. The general structure and main components of the VCI are described and related to our scenario. Finally, the chapter ends with the conclusion.

4.2 Street and Store Marketing Scenario (SSM)

Up-to-date ubiquitous computing with in-environment distributed sensors, actuators and user devices can create an Ambient Intelligence environment in a shopping area. The main goal is to detect potential shoppers and propose them appropriate goods, i.e. goods in relation to their shopping profiles and identified present needs. To do this, we need first to capture potential consumer presence by appropriate sensor(s), then to study his/ her profile and determine what kind of information it seems appropriate to present him/ her with in order to attract his/ her attention. The first stage in this capture process occurs in the street. Data collected by the sensors provide the system with information about the potential consumer, data that can be more or less precise: a young man or woman at least, or a store

regular client, etc. With respect to this profile, the system could display in the shop window an appropriate advertisement. In the shop window display, it seems important to display advertising information not only for one passer-by and potential client but for several. Display strategy can be organized by applying the proxemics user interface policy i.e. give more information to shop windows near located client(s) and less to distant ones. It may prove interesting to take two additional behaviors into account: in-shop continuation of consumer tracking, in order to provide more precise information in relation to his/ her movement in the store, and also in-the-street movement, in-the-street walking and shop window watching. In the first case, increasingly detailed information can be given to the potential consumer with respect to his/ her location in the store (suit, shirt, pants department, or kitchen furniture department) with detailed knowledge of his/ her situation (just married, etc.). The system can collect, store and use the information collected from previous purchases in the store or elsewhere (his/ her Facebook profile). In the second situation (in-the-street walking), it could be interesting to propagate the discovered profile of the potential consumer to other stores to allow them to use this information to provide him/ her with increasingly detailed and appropriate advertising.

4.3 Context-aware Middleware

Context-aware applications are becoming increasingly prevalent and can be found in the areas of wearable computing, intelligent environments, context-sensitive interfaces, etc. (Krumm, 2009). A now generally accepted definition of context, as mentioned earlier, is given by Abowd and Dey (Abowd et al., 1999): “Any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves”. The context-aware system is defined as the system that uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task.

Development of context-aware applications is inherently complex. These applications adapt to changed context information: physical context, computational context, and user context/tasks (Bettini et al., 2010). To achieve this aim, we need to integrate all sensors, actuators, communication objects and computing devices into the system. Low-level mechanisms and drivers are necessary. Then, either we have to create only one standalone application within which high-level context reasoning takes place, or we propose a set of applications or a system and propose creation of an application-independent common high-level of contextualization making it pos-

sible to collect, process, interpret and propagate information with the context model and reasoning mechanisms (Xu et al., 2011).

To implement this more in-depth approach to context-aware services in AmI, we designed a context-aware middleware, organized in two layers: the low layer is a sort of Enterprise Service Bus, which provides a solution to integrate sensors and actuators with a standardized data representation and unified standard interface to achieve the core functions of service interaction: service registry, service discovery, and service consumption.

The versatile context interpreter is our high layer, which is charged with context inferences, expressive query, and persistent storage. Detailed information will be provided in the following sections.

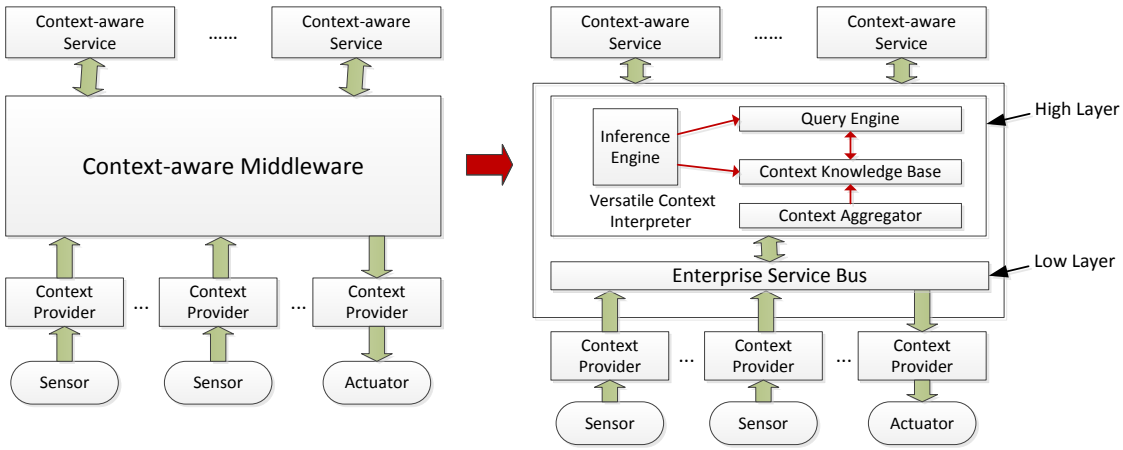


Figure.4.1 Two context-aware architectures

4.3.1 Low layer: Enterprise Service Bus

The low layer of our context-aware middleware is designed on a sort of ESB (Enterprise Service Bus), in charge of communicating in standard manner with in-environment sensors and actuators and of managing their dynamic discovery.

ESB is the increase prevalent structure in service-oriented architecture (SOA). Before detailing the ESB, we will review SOA. In a typical SOA environment, there is a service provider and a service consumer. In order to work, we also need a mechanism allowing them to communicate with each other and also describe and discover one another. The fundamental structure of SOA is shown in Figure.4.2. W3C has defined an open standard for web services in order to make SOA work. A web service uses the XML-based Simple Object Access Protocol (SOAP) for communication

between service provider and service consumer. The service provider describes the services as interfaces defined by Web Service Definition Language (WSDL). Universal Description, Discovery and Integration (UDDI), a platform-independent, XML-based registry, is used for looking up the services.

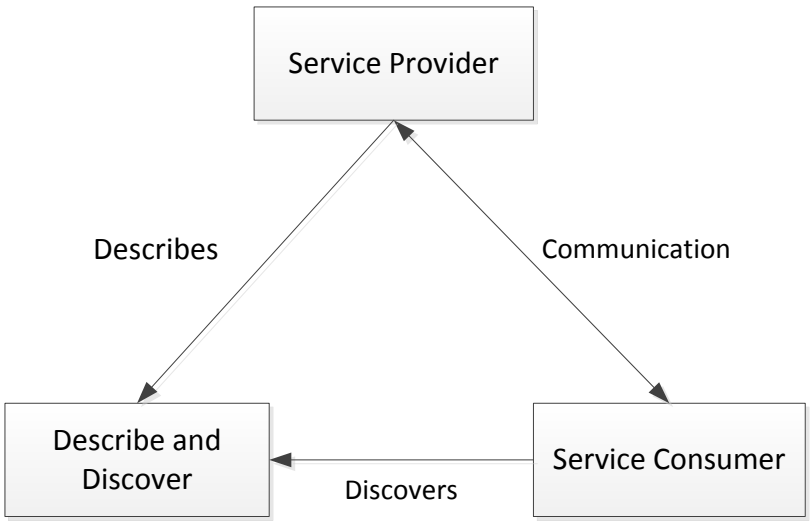


Figure.4.2 The fundamental structure of SOA

The first time the term “Enterprise Service Bus” appeared in the literature was in the book "The Enterprise Service Bus" by David Chappell (Chappell, 2004). The concept was developed in analogy to the Bus concept found in computer hardware architecture, and its goal is to describe implementation of loosely-coupled software components (known as services) that are expected to be independently deployed, running, heterogeneous and disparate within a network. It provides an infrastructure to implement this SOA concept. Instead of returning to point to point integration to implement SOA, ESBs extend knowledge gained in Enterprise Application Integration (EAI) work that simplifies the integration and flexible use of business components. ESBs facilitate standards-based integration, but are not limited only to web services-based integration, e.g. messages are not required to use only the http protocol for communication. It needs to be supported for transparent translation between communication protocols (e.g., HTTP, FTP, REST, SOAP, JSON, DCOM, CORBA, SAP RFC, etc.) A service consumer can send his/ her request via the http protocol, but the service provider provides data using JMS. The ESBs make this communication possible. In general, they support message-based transport as in EAI, and also provide a described and discover facility for the service provider and

service consumer. ESB allows services to be easily plugged in and out of the network without impact on other components and without the need to restart the system or even stop running applications. It is centered on a bus which provides a common backbone through which services can interoperate with a standardized format of data representation. ESB also provides mechanisms for security, mechanisms for implementing some level of service level agreements (SLA), routing and transformation capabilities, all of which vary to some extent from vendor to vendor.

Figure.4.3 The basic structure of ESB
(Image from (Bhat, 2008))

Consequently, ESB is an efficient and practical software architecture model used for designing and implementing interaction and communication between mutually interacting software applications in a service-oriented architecture (SOA). In our paper, it is used to connect VCI and the context provider, which is the service used to obtain context from sensors, the web or other sources, and dispatch commands to actuators. It helps the low layer to achieve three main tasks: communication, integration, and service interaction.

To better communicate with services is one of the major goals of ESB: communication support is thus necessary for ESB, which must deal not only with routing and locating services, but also with services concerning locating and naming. In communication processing, EBS allows many message delivery models, such as request/response, publish/subscribe, and

a large number of transport protocols as well as synchronous and asynchronous mechanisms.

ESB, via the bus mode, integrates all the services needed for context-aware application, enabling developers no longer to consider each other's message format or protocol between services, and improve system adaptability and scalability. The integrated services include context-aware providers, databases, service aggregation, and legacy systems, etc. Context-aware providers are related to various kinds of sensors and actuators developed in different programming languages (such as Java and c / c / c #) and in different application server environments (such as J2EE and. NET).

Service interaction is implemented in ESB when all the services are connected. It provides an open, implementation-independent messaging service and interface model, allowing the application code to be separated from the routing services and transport protocols. Service providers will post service description documents on the ESB platform for convenient access and use by service consumers. Furthermore, ESB helps to parse the message when required by the service.

The low layer, via ESB, provides a unified standard interface to achieve the core functions of service interaction: service registry, dynamic service discovery, and service consumption. It also integrates interaction devices and a set of APIs for different interaction modalities to support development of interaction approaches. In our SSM scenario, it manages all the context resources: physical sensors, such as: camera and Kinect; virtual resources: user profile, and sends these raw contexts to the high layer: the versatile context interpreter for the next process step.

4.3.2 High layer: The versatile context interpreter

The versatile context interpreter (VCI) is a high layer of context-aware middleware, as shown in Figure 4.1, made up of four parts: Context Aggregator, Inference Engine, Context Knowledge Base, and Query Engine. It leverages basic service results from the low layer to deliver and manage context-aware views and interpretations in order to deliver high-level information to the application.

As described in chapter three, our context-aware middleware adopts an ontology-based approach for context modeling and interpretation. General context-aware ontology can be completed and upgraded by more precise information related to a particular application or application area. In our case, the general context-aware model for AmI context-aware systems is considered to be the basic model. For a new application area such as the

"Street and Store Marketing" application (SSM application), we propose a more precise and specific context-aware model. According to our build methodology, the general context-aware model is developed as the whole system by context-aware middleware designers and developers. The "Street and Store Marketing" application is developed in the scope of the context-aware system by application developers. Most recent adaptation options can be implemented directly and dynamically during the application by the system (and its reasoning on collected data) or by the users (experienced users). According to different context sources, we divide context into two categories: the low-level context and the high-level context. Context that can be extracted directly from sensors and devices such as location, time, etc. is considered to be low-level context. High-level context is issued from inference treatment based on low-level context data and semantic rules stored in the knowledge base or using the inference engine.

After reviewing the context model used in our system, the detailed information of four parts in the VCI will be introduced as follows.

4.3.2.1 Context Aggregator

The context aggregator, a bridge between the low layer and the high layer, is responsible for working with basic contextual data collected by the low layer. Its structure diagram is shown in Figure 4.4. According to its function and position in the system, the context aggregator has one input and two outputs. It collects data from the low layer, converts them to OWL form and updates to the context knowledge base. Then this contextual information can be shared and reused by other service components. In addition, the context aggregator allows the VCI to send commands to the low layer.

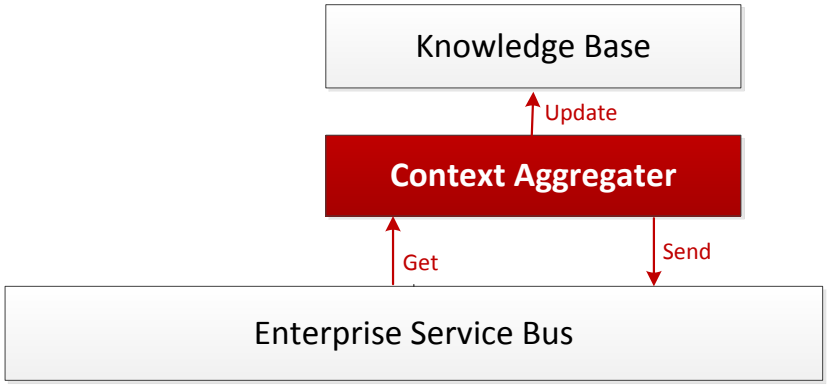


Figure.4.4 The context aggregator structure

Since contextual information is heterogeneous, one of the context aggregator's main tasks is to gather diverse raw information from the low layer in a single point. Each context aggregator component handles specific information source information respectively. Some of the context aggregator components work with the corresponding physical sensors in the smart space. For example, the location context aggregator is related to camera, QRcode, etc. while the proxemics context aggregator works with Kinect, used to measure the distance between device and end-users. The component can also extract contextual information from virtual sources. Users' schedules and profiles can be retrieved from the internet source by the virtual context components.

Another main task of the context aggregator is to convert raw contextual information extracted from low layers to information described in OWL format and to update to the context knowledge base. As contexts in the ubiquitous computing environment change frequently, the context aggregator needs to update regularly the Context Knowledge Base with new arrival contexts described in OWL. To ensure proper understanding, the location context aggregator is taken as an example. The location information gathered from sensors is converted to OWL-represented contextual information by the location context aggregator.

For example: "Tao locates in Shop" is represented in OWL format:

```
<Person rdf: about = "Tao">
        <locatedIn rdf: about = "Shop"/>
</Person>
```

Figure.4.5 The example represented in OWL format

Context aggregators are self-contained and self-configured components. Our context-aware middleware provides a universal interface for all context aggregator components. It enables a scalable and extensible context aggregator, thus simplifying development processing.

4.3.2.2 Context Knowledge Base

The context knowledge base provides persistent storage for context. As shown in Figure 4.6, the context knowledge base is located in the center of VCI, interacting with three other components. It has three inputs and one output from the inference engine, query engine and context aggregator, re-

spectively. Inputs from the inference engine and the context aggregator are used to add and to update context with proper date format to the knowledge base. The remaining inputs require execution by the query engine, helping context-aware applications find what they need. Its output returns the requested context to the context-aware application

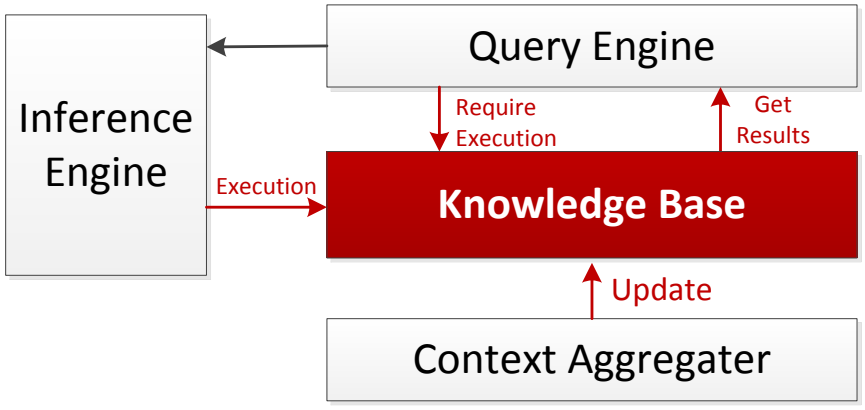


Figure.4.6 The knowledge base structure

The context knowledge base is different from the conventional relational database that purely supports storage and query. The main difference between database and knowledge base is in what is stored. We adopted the aforementioned ontology-based context model to build the environment and the user model. In ontology, the context knowledge base is made up of ABox and TBox. ABox and TBox are used to describe two different types of statements. TBox statements describe a conceptualization, a set of concepts, and properties for these concepts ("Tbox," 2009). ABox are TBox-compliant statements about individuals belonging to those concepts ("Abox," 2009). In OWL, ABox refers to Scheme, and TBox refers to Data. We use Figure 4.7 to show the relationships between OWL and the knowledge base.

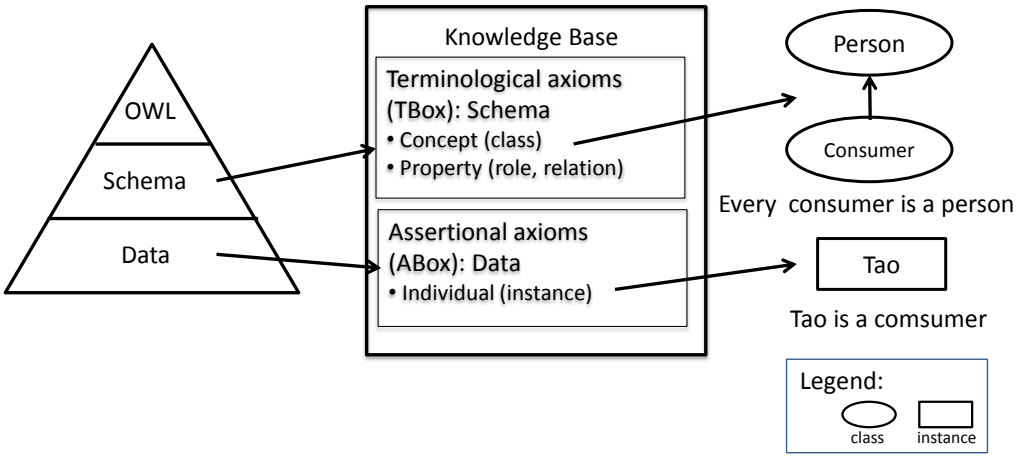


Figure.4.7 The relationship between knowledge base and OWL

The entire context is stored as the triple pattern including subject, predicate and object. It refers not to the data but the relationship, which can characterize the entity's situation to answer "who", "when", "what" and "where" of context, for example "where is the customer?", "who is in the shop?" and "when will the activity start" Besides storing the context, the ontology-based context model paves the way for the inference engine.

Another function: the Context Knowledge Base supplies a set of library procedures for other components to query and modify context knowledge as well. In relation to different sources, the context is divided into three categories: pre-defined context, detected context, and inferred context. Pre-defined context refers to context expressed in the application context model elaborated by application designers, such as user's profile context and specific environment context. Detected context is obtained from sensors, viewed as low-level context. Inferred context is determined from collected data and knowledge rules by inference engine, and is considered as high-level context. In order to facilitate development, we have defined six types of models to deal with different situations in application, as shown in Figure 4.8.

```
public class ContextModel {


   final public String ex = "http://liris.cnrs.fr/smartspace.owl#";

   private ModelChangedListener forAdding;

   /** The model that stores context ontology information */
   private OntModel contextOntology;
   /** The model that stores static context data */
   private OntModel staticContexts;
   /** The model that stores historical sensed context data */
   private OntModel storedSensedContexts;
   /** The model that stores sensed context data */
   private OntModel sensedContexts;
   /** The model that stores all context data */
   private OntModel contextData;
   /**
    * The inference model containinig all context data, with further RDFS
    * reasoning performed on it
    */
   private InfModel baseModel;
```

Figure.4.8 The partial knowledge base code

We design API for updating context from different resources as appropriate. Context-aware application developers can use them to modify ontology class, individuals and properties. The fundamental function, as an example, is provided in Figure 4.9, and is used to add new context to the ontology-based model.

```
public static OntModel addStatementToModel(Statement s, OntModel m)
{
    if (!m.contains(s.getSubject(), s.getPredicate(), s.getObject()))
    {
        // Remove all statements with this subject-predicate pair
        //System.out.println("Removing existing context data");
        Model tempModel = ModelFactory.createOntologyModel();
        tempModel.add(m);
        StmtIterator sti = tempModel.listStatements(
                s.getSubject(),
                s.getPredicate(),
                (RDFNode)null);
        for (Iterator it = sti; it.hasNext(); )
        {
            Statement st = (Statement)it.next();
            // Remove this statement
            m.remove(st);
        }
        tempModel.close();
    }

    m.add(s);
    System.out.println("[ContextManager] - Statement added");
    return m;
}
```

Figure.4.9 One of the fundamental functions in the knowledge base

4.3.2.3 Context Query Engine

The context query engine is used to handle expressive queries from context-aware applications and to respond to context-aware applications with up-to-date query results. It is the main component for communicating context-aware service in the VCI. Its structure diagram is shown in Figure 4.10. The crucial input is from the context-aware service for sending queries to the knowledge base, while the crucial output also uses the context-aware service to obtain results from the knowledge base. Besides the basic function, it can be also used to invoke the context inference engine. Two main tasks are explained below:

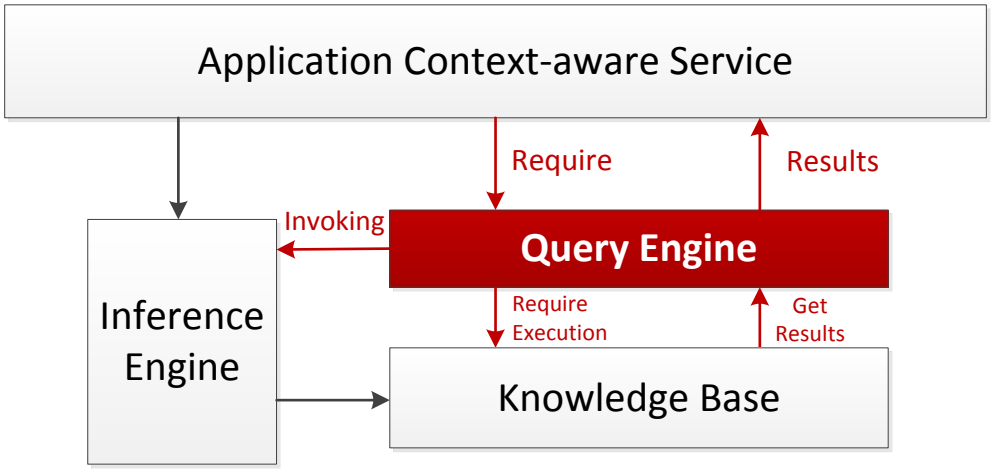


Figure.4.10 The query engine structure

Handling queries from the context-aware application is the main task of the context query engine. As mentioned earlier, our context-aware middleware uses the knowledge base to store context in OWL format: its query format is also different from that of the conventional SQL. We use a query language, SPARQL, to retrieve and handle data stored in the context knowledge base in OWL format. SPARQL, a recursive acronym for SPARQL Protocol and RDF Query Language, is a standardized query language for RDF data with multiple implementations, offering developers and end-users a way to write and consume the results of queries across this wide range of information.

For different purposes, the SPARQL language specifies four different query variations for queries that read data from the database.

- **SELECT query:** used to extract raw values from a SPARQL endpoint, the results are returned in a table format.

- **CONSTRUCT query:** used to extract information from the SPARQL endpoint and convert the results into valid RDF.

- **ASK query:** used to provide a simple True/False result for a query on a SPARQL endpoint.

- **DESCRIBE query:** used to extract an RDF graph from the SPARQL endpoint, the contents of which are left to the endpoint to decide, based on what the maintainer considers to be useful information.

Each of these query forms takes a WHERE block to restrict the query, although, in the case of the DESCRIBE query, the WHERE is optional.

The notable advantage of SPARQL is that it allows a query to consist of triple patterns, conjunctions, disjunctions, and optional patterns. It is thus recognized as one of the key technologies of the semantic web, becoming an official W3C Recommendation on 15th January 2008 ("SPARQL Query Language for RDF," 2008). It allows the context-aware application to:

- Pull values from structured and semi-structured data.
- Explore data by querying unknown relationships.
- Perform complex joins of disparate databases in a single, simple query.
- Convert RDF data from one vocabulary to another.

A simple example is shown in Figure 4.11, which helps understand SPARQL language. Its aim is to find "who is in Shop XXX?" A result clause followed by the keyword "select" identifies the information to be returned from the query. The query pattern in the brace after the keyword "where" specifies what to query for in the underlying dataset.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ex: <http://liris.cnrs.fr/smartspace.owl#>

select ?Who

where {?Who
        ex:locatedIn
        ex:Shop XXX }
```

Figure.4.11. The example for query in SPARQL format

In reality, we define several functions for handling queries from context-aware applications. The two basic functions are shown in Figure 4.12 as an example. Function one, shown in Figure 4.12(a), aims at reading query information from the target resource and returning a processable string. Function two, shown in Figure 4.12(b), aims at running a query for the ontology model and returning the results.

```java
/** Read query information from target file and return query string back */
public String readQuery(String targetFile) throws IOException{

    // Read query information from target file and return query string back */

    BufferedReader reader = new BufferedReader(new FileReader(targetFile));
    int readNumber = 0;
    char[] tA = new char[1024];
    StringBuffer queryBuffer = new StringBuffer(1000);
    String qString;

    // Parse query string
    try {

        while ( (readNumber = reader.read(tA)) != -1){

                qString = String.valueOf(tA, 0, readNumber);
                queryBuffer.append(qString);
                tA = new char[1024];
        }

        reader.close();


    } catch (FileNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    // Return query string
    return queryBuffer.toString();

}
```

(a) The read query function

```java
/**Run query for the model*/
public void runQuery(String queryString,Model model){

    // build tempearyy query
    Query query = QueryFactory.create(queryString);

    //run query for model
    QueryExecution qe = QueryExecutionFactory.create(query, model);

    //get results
    ResultSet results = qe.execSelect();

    //format and pass the result
    ResultSetFormatter.out(System.out,results,query);

    // release the query
    qe.close();
}
```

(b) The model run query function

Figure.4.12 The Query Engine functions

The Context Query Engine also has another function: invoking the context inference engine. When the application needs high-level context, it provides a method to invoke the context inference engine to generate the inferred context. The partial code is shown below, trying to invoke the inference engine according to the query string.

```
// Read Query File
QueryEngine nQuery = new QueryEngine();

queryString = nQuery.readQuery("smartspace.query");

// Find key word
MatchQueryKeyWord nMatch = new MatchQueryKeyWord();

queryKeyWord = nMatch.getKeyWord(queryString);


// Invoke the inference engine
OntologyReasoner ontoReasoner = new OntologyReasoner();

nameRuleFile = ontoReasoner.chooseReasonRule(queryKeyWord);

infOntModel = ontoReasoner.runEngine(nameRuleFile, model);
```

Figure. 4.13 The partial code for invoking the inference engine

4.3.2.4 Context Inference Engine

The context inference engine is an important part of the context-aware system. It is the intelligent component in context-aware middleware, aiming at deducing high-level context from low-level context retrieved directly from context resources, as well as checking context consistency when new context arrives. As shown in Figure 4.14, context-aware services are able to handle the inference engine directly. Inference engine functions depend on the situations, invoked by different mechanisms (Is this the meaning here?). The inferred results will be added to the knowledge base, and used by other components.

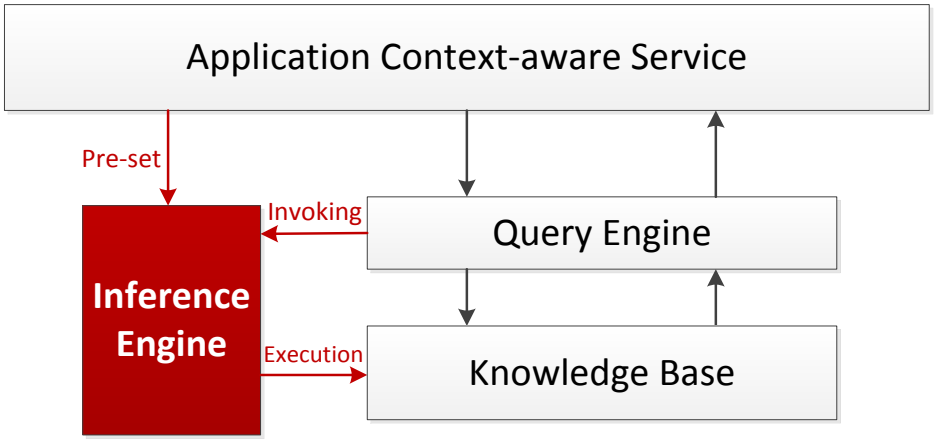


Figure.4.14. Inference engine structure

The context inference engine consists of two modules: the basic inference module and the predictive module. These modules are responsible for two main tasks of the context inference engine, shown in Figure 4. 15.

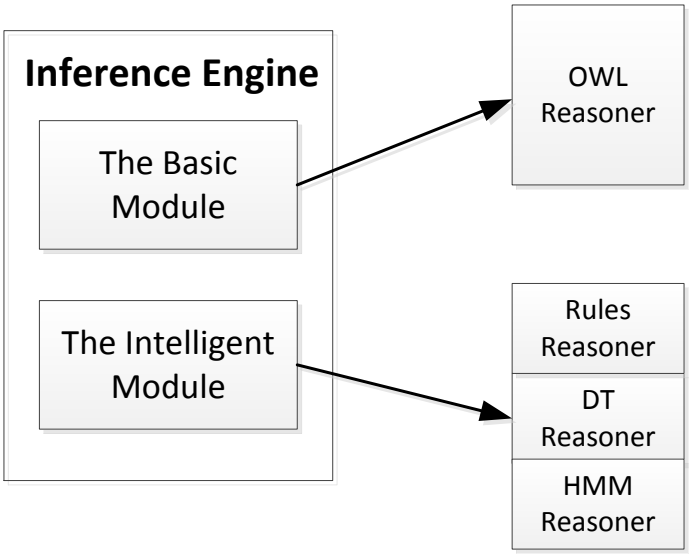


Figure.4.15. The two different inference modules and their use

The basic inference module consists of the OWL Reasoner. Its main task focuses on checking context consistency. Consistency checking of the context model is an important action, activated by changing contextual information or by the developer modifying the context-aware basic model. The OWL Reasoner is a built-in reasoner in OWL language, performing competently for this purpose. Based on the ontology model, it provides a set of described logic rules to accomplish this task.

The intelligent module consists of three reasoners: Rules Reasoner, Decision Tree (DT) Reasoner, and Hidden Markov Models (HMM) Reasoner. It is in charge of the other main task performed by the context inference engine: inferring high-level context. High-level context refers to the type of context that cannot be collected directly from context resources. In our studies, we investigate activity context. According to analysis of activity context recognition studies over the last decade, we summarize activity context into three facets: basic activity inference, dynamic activity analysis, and future activity recommendation. These three reasoners help the context inference engine meet the requirements of the three facets of activity context recognition.

The Rules reasoner works on basic activity inference based on first-order logic rules. Context-aware application designers could freely define different rule sets to deal with different basic activities in smart spaces, such as meeting, working, studying, and so on.

The HMM reasoner is designed to solve the problems of dynamic activity of analysis. In HMM, dynamic activity is divided into a serious continues slices. After the training process, it infers on-going activity by calculating the probability model of activity.

The DT reasoner provides a practical solution for future activity recommendation based on a decision tree algorithm. According to analysis of the user's previous activities, the decision tree can be built to provide the customized recommendation for an individual.

The three reasoners provide a solution for three facets of activity context recognition, respectively. In the inference engine, different reasoners have different invoking mechanisms to make them work at the right time. Once the reasoner is invoked, the knowledge base should be updated accordingly. This enables the context-aware application to provide coordinated services according to inferred results. The context inference engine is one of my main contributions to this thesis and will be detailed in the next chapter.

4.3.3 Context-aware middleware behavior workflow

To summarize our proposal of a user-centric context-aware middleware in ambient intelligence, we comment on the global workflow as shown in Figure 4.16: in the overall architecture we have 3 main components: an application layer with high-level context-aware services related to the application, a high-level middleware known as VCI (Versatile Context Interpreter), and a low-level middleware based on an ESB (Enterprise Service Bus). As

stated earlier, two levels of behavior and their modeling are supported: a basic level related to general problems of AmI and context modeling, and a more specific level related to an application area. In particular, the application layer informs the VCI layer of the application area to take into account, while the ESB layer is mainly application area independent. With this in mind, the overall workflow functions as follows: (1) The application must inform the VCI of the context (specific model) to use. This model will be used by the Inference Engine, Query Engine, Context KB, and Context Aggregator. (2) Application Context-Aware Services ask to receive contextual evolution from the VCI. (3) ESB collects the data from different sensors and propagates them to the context aggregator. (4) When the Context Aggregator is able to aggregate the received data, it does so, and places them in Context KB. (5) Arrival of new data in Context KB generates the notification to the Context Query Engine. (6) The Query Engine calls on the Context Inference Engine to apply context inference. (7) The Context Inference engine introduces inferred data to Context KB. (8) When the Inference Engine terminates the inference process, the Context Query Engine collects new data from Context KB. (9) The Context Query Engine sends these data to the application. (10) The application context-aware services can also decide to update actuator states. They send new data to the Context Aggregator, which propagates the data to the appropriate actuator using ESB.

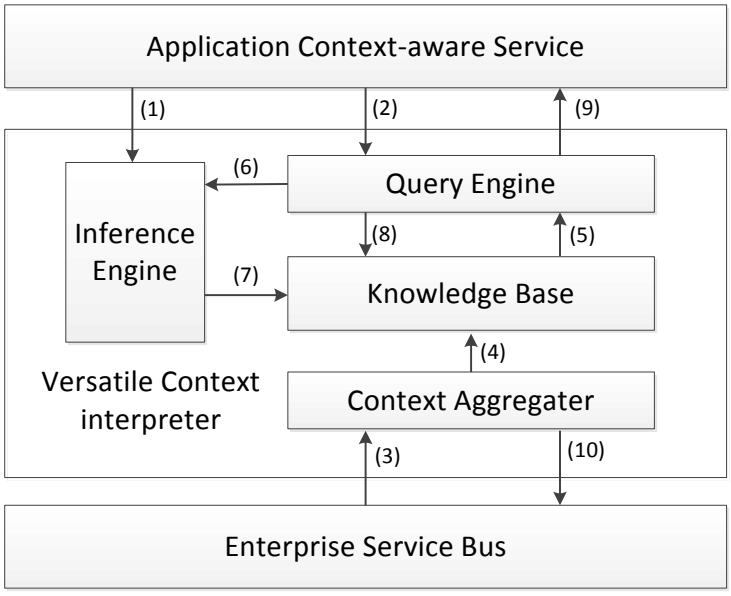


Figure.4.16. Context-aware middleware behavior workflow

4.4 Conclusion

Nowadays, context awareness has become an increasingly prevalent term in the area of ambient intelligence. This concept is widespread and implemented in a variety of fields. And yet development of context-aware applications is still a tough task. In this chapter, context-aware middleware is proposed as an efficient solution. As context resources (physical sensors and virtual services) become increasingly diverse and complex, manageability is frequently a concern. To limit difficulty of context-aware application development, we adopt two-layer structure middleware to deal with context collecting and context processing, respectively.

The low layer, based on EBS, provides a solution to communicate in standard manner with in-environment sensors and actuators and a unified standard interface to achieve the core functions of service interaction: service registry, service discovery, and service consumption.

The high layer is the versatile context interpreter, consisting of four parts: Context Aggregator, Inference Engine, Context Knowledge Base, and Query Engine, dealing with context inferences, persistent storage, and expressive queries.

This platform provides methods for solving the issues relating to service discovery, mobility, and environmental changes. It also provides a rapid prototyping of new context-aware applications in ambient intelligence. In addition, to ensure proper understanding, the chapter provides a Street and Store Marketing scenario at the beginning, and context-aware middleware behavior workflow at the end.

The content of this chapter was published in:

Xu, T., Hui, L.J., David, B., Chalon, R., Zhou, Y., A context-ware middleware for interaction devices deployment in AmI, the 15th International Conference on Human-Computer Interaction (HCI'13), Las Vegas, Nevada, USA

Xu, T., David, B., Chalon, R., Zhou, Y., A Context-aware Middleware for Ambient Intelligence. ACM/IFIP/USENIX 12th International Middleware Conference, Lisbon, Portugal. Middleware Posters'2011

5 The Intelligent Inference Engine

5.1 Introduction

The context-aware system is an answer to challenges associated with service discovery, mobility, environmental changes, and context retrieval (Romero et al., 2008). Context inference plays the role of the brain in the context-aware middleware, which contains two main tasks: one is checking context consistency; the other is determining or inferring the user's situation. Once the user's situation has been inferred, the application can take an appropriate action (Krumm, 2009). In recent years, most context-aware middleware has adopted the ontology-based model. In this model, OWL is responsible for checking context consistency. The research focuses are shifted to infer the user's situation (commonly referring to activity). The existing context-aware middleware usually employs the rule-based method to solve this problem. This method is relatively easy to build and relatively intuitive to work with. However, it is fragile, not flexible enough, and is inadequate to support diverse types of tasks. The holy grail of context awareness is to divine or understand human intent (Krumm, 2009). It requires a much smarter "brain" to deal with various issues. Based on this point, context inference is expected to support versatile activity context recognition in context-aware computing.

The Active Badge System (Want et al., 1992) is commonly considered to be the first research investigation into context awareness. In this work, context information refers to location primarily. From then on, numerous infrastructures have provided services for handling context. The

context inference part always plays an important role in these systems. The Context Toolkit, developed by (Dey et al., 2001), is a toolkit supporting the development of context-aware applications. A context interpreter is used to deduce the high-level context information from the low-level information. CASS (Fahy and Clarke, 2004) supports context-aware applications on hand-held computing devices, and other small mobile computing devices. The important feature of CASS is that it supports to abstract the high-level context and separate context based on inferences and behaviors from the application code. CoBrA (Chen et al., 2003) is one of the earliest systems using semantic web technology to support context-aware pervasive computing. The inference engine of CoBrA is used in the two types of reasoning. Besides detecting and resolving inconsistent knowledge, it can also infer context knowledge that cannot be easily acquired from the physical sensors. The context reasoner in SOCAM (Gu et al., 2004) supports two kinds of reasoning: ontology reasoning and user-defined rule-based reasoning. Truong and Dustdar have summarized inference technique support in the existing system (Truong and Dustdar, 2009). However, they found that context inference is inadequate and that most systems are just based on semantic reasoning.

With the emergence of various location sensors, “location” is no longer the context with which researchers’ are most concerned, and is being gradually replaced by "activity". Wang et al. leverage rules based on first-order logic to infer user’s activity (Wang et al., 2004). To better recognize user’s activity, a large number of artificial intelligence methods are used in context-aware computing. Panu Korpipaa et al. (Korpipaa et al., 2003) use a naïve Bayes classifier to recognize higher-level contexts from lower-level contexts. The Hidden Markov Model is employed to recognize activities in the work of (van Kasteren et al., 2008). Pollack et al. use the decision tree to make decisions about whether and when it is most appropriate to issue reminders for prescribed activities (Pollack et al., 2003). However, these approaches only provide a possibility to solve part of the problem in activity context recognition, and none of them takes into account how to integrate these approaches in the context-aware system and work well together with other parts of the system.

In this chapter, we review all the methods on activity context recognition published in three premier conferences in the past decade, and conclude that activity context recognition is divided into three facets: basic activity inference, dynamic activity analysis, and future activity recommendation. Then we propose an intelligent inference engine based on our context-aware middleware. This integrates the three most popular methods of activity context recognition used in context-aware application, and pro-

vides a solution to meet different requirements in activity context recognition.

5.2 Context and Activity Context Recognition

Context is one of the critical concepts in context-aware computing. With respect to the latter, they focus on who, where, when and what (that is, what activities are occurring) of entities and use this information to determine why a situation is occurring (Krumm, 2009). The research work on context awareness is simplified to achieve the five "w". Among these "w", the "why" is the destination, and the other four "w" (who, where, when, what) represent four kinds of fundamental context: user, location, time, and activity. With the rapid development of technology and widespread use of smart mobile devices and sensors, the first three fundamental contexts (user, location and time) are now relatively easily captured directly from the context sources. However, recognizing user's activities is still a tough task.

Activity context recognition aims to infer a user's behavior from observations such as sensor data (Hu et al., 2011), and has various applications including medical care (Pollack et al., 2003), logistics service (Lin, 2006), robot soccer (Vail et al., 2007), plan recognition (Geib et al., 2008), etc. Many researchers make great attempts to find the most efficient way to recognize user's activities. Continuing Lim and Dey's work (Lim and Dey, 2010), we reviewed literatures from three top-tier conferences on context-aware computing in recent years: the ACM Conference on Human Factors in Computing Systems **(CHI)** from 2003 to 2012, the ACM International Conference on Ubiquitous Computing **(Ubicomp)** from 2004 to 2012, and the International Conference on Pervasive Computing **(Pervasive)** from 2004 to 2012. There are 59 papers related to user's activity recognition, which involve seven algorithms: Rules, Decision Tree (DT), Naïve Bayes (NB), Hidden Markov Model (HMM), Support Vector Mechanic (SVM), k-Nearest Neighbor Algorithm (kNN), and Artificial Neural Networks (ANNs). The detailed statistical results are shown in Figure.5.1.

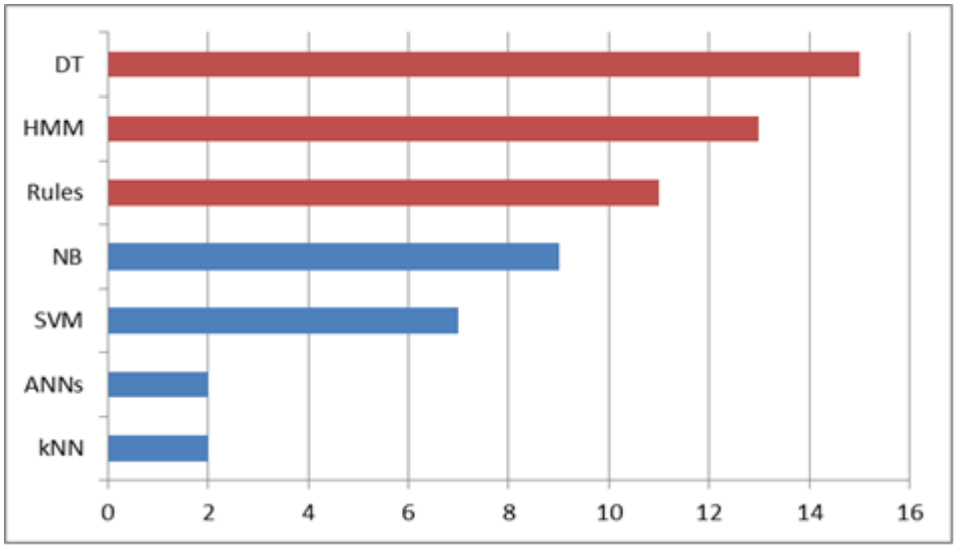


Figure.5.1. The Statistical Results of the Activity Recognition Algorithm

We conclude research on recognizing user's activity in three facets shown in Figure 5.2: basic activity inference, future activity recommendation, and dynamic activity analysis.

- **Basic activity inference** focuses on recognizing the basic types of activity, such as working, sleeping, and shopping. User's activity can be determined directly based on low-level context obtained from diverse context sources containing physical context sensors and virtual context source (web service). The method based on Rules is used to infer basic activity.
- **Future activity recommendation** usually refers to recommending or predicting the user's future activity or choices. To this aim, besides taking into consideration the user's current situation (context), it requires analyzing the context of the user's previous similar behaviors (training data). The decision tree is adequate to tackle this problem. It can be viewed as a type of context-aware recommendation.
- **Dynamic activity analysis** is responsible for analyzing fine-grained activity compared with basic activity inference. There are two main solutions: one employs the State-Space Model, which partitions the activity into state sequences and infers the type of activity (state sequence) based on probability of an observed sequence such as Bayesian Networks and the Hidden Markov Model. The other relies on pattern recognition techniques, which extract patterns from activities and infer the type of activity based on different activity patterns, like the Vector Support Machine, Artificial Neural Network, etc. As it is similar to the future activity recommendation, it also

requires learning the process parameters from the previous activity information.

It can be easily observed that the top three most popular methods are Rules, DT, and HMM. Each method has its own advantages and characteristics. We analyzed them (the detailed information will be provided in the following section) and found that the three methods can be leveraged to deal with these three facets respectively. Also, these three methods mostly represent the mainstream solution on their actual facets. We integrate the three methods, Rules, DT, and HMM, into our context-aware middleware as our research goal.

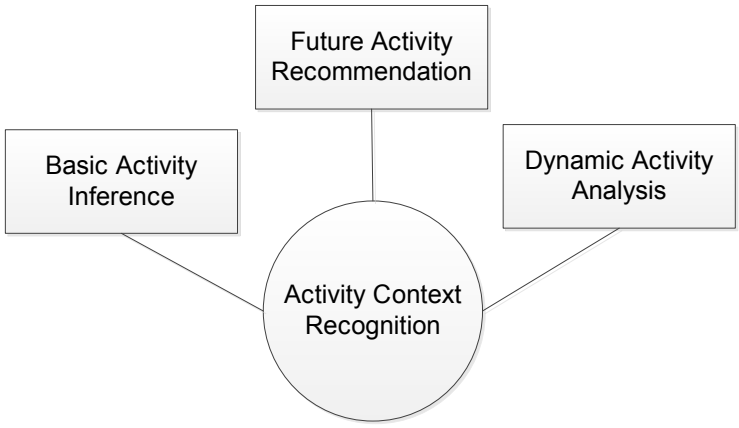


Figure 5.2. The Three Main Facets of Activity Context Recognition

5.3 The Intelligent Inference Engine

The current user's context is made up of an undefined number of contextual values. Each value is described by two elements: an unambiguous ID and a probability value. We divide contextual values into two categories: concrete contexts and abstract contexts. Concrete contexts, also considered as low-level context, represent the information obtained by a set of sensors. These contexts can be read from the surrounding environment through physical sensors (e.g., temperature sensor) or can be obtained by other software (e.g., calendar) through logical sensors. Some examples are: "temperature: 20°C," "12:30," "meeting at 14:30" and so on. Concrete contexts are returned by the sensors and represent the input of the inferential mechanism. Abstract contexts, also viewed as high-level context, represent everything that can be inferred from concrete contexts, for example, "user at home," "user is shopping," etc. The problem we are facing is, therefore, the definition of an inferential system able to derive the abstract contexts

from the concrete contexts. Concrete and abstract contexts are the inferential system input and output, respectively. From a theoretical point of view, this difference is small since the contexts cannot be unambiguously assigned to one or the other category: the context "temperature 45 °C" can be a concrete context as it is obtained from a sensor or it can be inferred by other contexts (e.g., "user in sauna"). The aim of the inferential system is to combine concrete contexts to determine abstract contexts and to combine abstract contexts to obtain new, more abstract contexts. The purpose of our study is the activity context.

5.3.1 Basic Structure of the Intelligent Inference Engine

For a better understanding of human intent, we propose an intelligent inference engine for our context-aware middleware, consisting of a basic module and an intelligent module as shown in Figure 5.3. Besides satisfying the basic requirements, it can provide appropriate methods for different facets of activity context recognition.

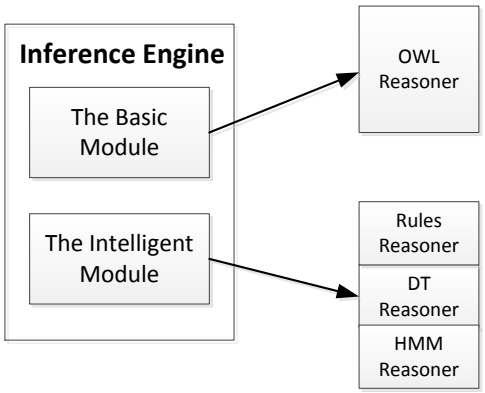


Figure.5.3 The Inference Engine structure

5.3.1.1 The basic module

The basic module's main task is checking context consistency. We adopt the ontology-based context model to construct our context-aware system. This model has good features for developing context-aware systems, such as knowledge sharing, knowledge reuse, and logic inference. Also it paves the way for the basic inference module**.** For the context model, we employ a hierarchical structure to describe the user's situation and circumstance based on OWL. This structure is shown in chapter 3. In ontology, the context knowledge base is made up of ABox and TBox. As described in chap-

ter 3, ABox and TBox are used to describe two different types of statements. TBox describes a conceptualization, a set of concepts, and properties for these concepts, while ABox concerns instances belonging to those concepts. In OWL, ABox refers to Data, and TBox refers to Scheme. The context knowledge base enables the reasoner to work. Consistency checking operations check that instances of data are consistent with a given schema.

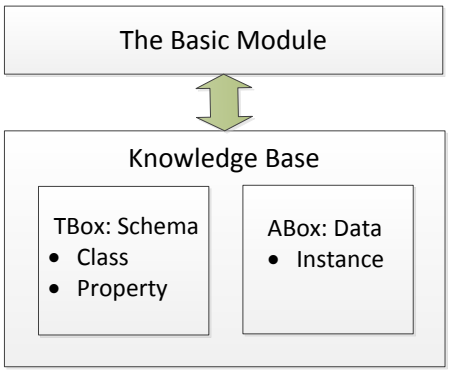


Figure.5.4 The relationship of the knowledge base inference engine

OWL has a built-in reasoner based on the description logic. This reasoner can satisfy most logical requirements, which comprise class subsumption, class consistency, concept satisfaction ability and instance checking.

- Check the consistency of the ontology and knowledge: assume we have declared x to be an instance of class A. Further assume that
  - A is a subclass of B⋂C
  - A is a subclass of D
  - B and D are disjoints

  Then we have an inconsistency because A should be empty, but has the instance x. This is an indication of an error in the ontology.

- Check for unintended relationships between classes. Class membership: if x is an instance of a class C, and C is a subclass of D, then we can infer that x is an instance of D. Equivalence of classes: if class A is equivalent to class B, and class B equivalent to class C, then A is equivalent to C, too.

- Classify instances in classes. If we have declared that certain property-value pairs are a sufficient condition for membership of a class A, then if

an individual x satisfies such conditions, we can conclude that x must be an instance of A.

The basic module enables developers to design large ontologies, where multiple authors are involved, and to integrate and share ontologies from various sources. Besides context consistency, it also provides functions concerning properties and their values. The figure below shows a part of basic reasoning properties. In addition, it helps facilitate development of context-aware applications.

Property	Function
Transitive Property	P(x,y) and P(y,z) implies P(x,z)
Symmetric Property	P(x,y) iff P(y,x)
Functional Property	P(x,y) and P(x,z) implies y = z
inverseOf	$P_1(x,y)$ iff $P_2(y,x)$
Inverse Functional Property	P(y,x) and P(z,x) implies y = z

Figure.5.5. The partial basic reasoning properties for OWL

5.3.1.2 The intelligent module

In our intelligent inference engine, the intelligent module is responsible for supporting activity context recognition. According to aforementioned reviews on methods of activity recognition, we integrate the most popular methods, Rules, DT, and HMM, to provide a solution for three facets of activity context recognition: basic activity inference, dynamic activity analysis, and future activity recommendation.

5.3.1.2.1 Rules

"Rules" is the early attempt to infer user's activity in context-aware computing. "Rules" refers to the first-order logic, also known as first-order predicate calculus or first-order functional calculus, symbolized reasoning in which each sentence, or statement, is broken down into a subject and a predicate. For activity context recognition, "Rules" is a method using a set of "if-then rules" to infer user's activity: if the devices sense a particular situation, they can deduce user's activity. "Rules" is based on general features of activity. These chosen features of activity, namely low-level context, should be able to be obtained by physical sensors or other context sources.

It contains two main methods of reasoning with inference rules: forward chaining and backward chaining. Forward chaining starts with the

available data and uses inference rules to extract more data until a goal is reached. In contrast, backward chaining starts with a list of goals and works backwards from the consequent to antecedent to see if there is data available that will support any of these consequences. We try to use a simple example to explain both of them.

There is an existing knowledge clause, using "if-then" representation. Also, two known facts are provided. They are shown below:

Knowledge clause:

If X is Y's parent and a male, then X is Y's father.

The known facts:

Ming is Tao's parent.

Ming is a male.

Figure.5.6. The sample knowledge clause and facts

With forward chaining, it can infer that "Ming is Tao's father" in three steps:

1. Ming is Tao's parent **and** Ming is a male.
2. Based on logic, it can infer: Ming is Tao's parent **and** a male.
3. Based on rules, it can infer: Ming is Tao's father.

The detailed pseudo code of the forward chaining method is provided in Figure 5.7.

Algorithm Forward-Chaining-Method

Inputs: *KB*, a set of first-order definite clauses
α, the query, an atomic sentence
Local variable: *new*, the new sentences inferred on each iteration
Output: *a substitution or false*

Repeat until *new* is empty
new ← {}
for each sentence r **in** KB **do**
$(p_1 \wedge \ldots \wedge p_n \rightarrow q) \leftarrow \text{STANDARDIZE} - \text{APART}(\gamma)$
for each θ such that $\text{SUBST}(\theta, p_1 \wedge \ldots \wedge p_n) = (\theta, p'_1 \wedge \ldots \wedge p'_n)$
for some $p'_1 \ldots p'_n$ in KB
$q' \leftarrow \text{SUBST}(\theta, q)$
if q' is not a renaming of some sentence already in KB or new **then do**
add q' to new
$\phi \leftarrow \text{SUBST}(\alpha, q)$
if ϕ is not fail **then return** ϕ
add *new* to KB
return *false*

Figure.5.7. The pseudo code of the forward chaining method

With backward chaining, it can answer the question "Who's Tao's father ?" according to the knowledge cause and facts mentioned in Figure 5.8. in four steps:

1. "?" is Tao's father.
2. Based on rules, it can derive: "?" is Tao's parent and a male.
3. Based on logic, it can infer: "?" is Tao's parent and ? is a male.
4. Based on the facts, it can derive: Ming is Tao's parent and Ming is a male.

This derivation will cause it to produce Ming as the answer to the question "Who is Tao's father?".

The detailed pseudo code of the backward chaining method is provided in the Figure below.

Algorithm Backward-Chaining-Method

Inputs: *KB*, a set of first-order definite clauses
$goals$, a list of conjuncts forming a query(θ already applied)
θ, the current substitution, initially the empty substitution{}

Local variable: *answers*, a set of substitutions, initially empty

Output: *a substitution*

If *goals* is empty **then return**{}
$q' \leftarrow$ SUBST(θ, FIRST(*goal*))
for each sentence r **in** *KB* where STANDARDIZE − APART(γ) → ($p_1 \wedge ... \wedge p_n \rightarrow$ q)
add $\theta' \leftarrow$ UNIFY(q, q') succeeds
new_goals ← [$p_1 ... p_n$ | REST(*goals*)]
answers ←Backward-Chaining-Method (KB, *new_goals,* COMPOSE(θ, θ'))
Return *answers*

Figure 5.8. The detailed pseudo code of the backward chaining method

Rules require sentences to be in a standard form, known as Horn clauses named after the mathematician Alfred Horn (Horn, 1951). A Horn clause is a sentence of the form:

$$q_1 \ and \ q_2 \ and \ ... \ and \ q_n \ -> \ r$$

where each q_n and r is an atomic sentence and all variables are universally quantified. In implementation, there is a similar format with the

Horn clause, which can work with OWL in Jena. We employ a simple example as shown below to interpret how to infer a person is working with machine-readable format. It images a scenario on working in lab, which means that "If the current time is after 9 am, the person is in lab, lab light is on and lab door is open, then it can infer that the person is working."

```
@prefix ex: <http://liris.cnrs.fr/smartspace.owl#>.
[Working:
   (?Event ex:has ?time) greaterThan(?time,9)
   (?person ex:hasLocatedIn ex:lab)
   (?light ex:hasStatus ex:ON)
   (?door  ex:hasStatus ex:ON)
->
   (?person ex:isDoing ex:work) ]
```

Figure.5.9. This is an Example of Rules

"Rules" has a wide range of adaptabilities, which is usually designed to provide inference for almost all users. Furthermore, it is relatively intuitive and easy to work with. However, rules are rigid, meaning that they are also brittle (Krumm, 2009). Even trifling exceptions will cause some errors. Since this method is easy to be implemented in context-aware applications, it is an effective and widespread method used to infer user's activity. We leverage it to work on basic activity inference.

Besides Rules, the common alternative approaches involve artificial intelligence algorithms. DT and HMM both belong to this category. Each has its own features, focusing on different facets.

5.3.1.2.2 Decision Tree (DT)

The Decision Tree is a classic algorithm in the field of artificial intelligence. It is a predictive model which maps observations on an item to conclude as to the item's target value. Decision trees are learned by recursively partitioning training data into subgroups until those subgroups contain only instances of a single class. The processing for partitioning data runs on the values of item attributes. The choice of the item attribute on which to operate the partition is generally made according to the entropy criterion and the information gain. The entropy of S can be described below, which is a measurement of the expected encoding length measured in bits.

$$Entropy(\mathrm{S}) = \sum -p(\mathrm{s}) \log_2 p(\mathrm{s})$$

The information gain is the expected reduction in entropy caused by partitioning the examples according to this attribute. The information gain, Gain(S, A) of an attribute A, relative to a collection of examples S, is defined as:

$$Gain(\mathrm{S}, \mathrm{A}) = Entropy(\mathrm{S}) - \sum_{v \in Value(\mathrm{A})} \frac{|S_v|}{|S|} Entropy(\mathrm{S}_v)$$

where Values (A) is the set of all possible values for attribute A, and is the subset of for which attribute A has value. The information gain is used to select the best attribute at each step in growing the tree.

We provide the pseudo code to show implementation of the decision tree in Figure 5.10.

```
Algorithm Decision-Tree returns
  Inputs: examples, set of examples
          attribs, set of attributes
          default, default value for the goal predicate
  Output: a decision tree
if examples is empty then return default
else if all examples have the same classification then return the classification
else if attribs is empty then return Majority-Value(examples)
else
    best <- Choose-Attribute(attribs, examples )
    tree <- a new decision tree with root test best
    m <- Majority-Value(examples)
    for each value v of best do
                examples_i <-{elements of examples with best = v_i}
                subtree <- Decision-Tree(examples_i, attribs - best, m)
                add a branch to tree with label v_i and subtree sub-tree
    return tree
```

Figure 5.10. The pseudo code of the Decision Tree

The Decision Tree is simple to understand and interpret. Users are able to understand decision tree models after a brief explanation. More clearly, the classic example is shown in Figure 5.11. The example shows whether to play tennis depending on the weather condition. Decision trees are popular for their simplicity of use, interpretability, and good runtime performance. They are commonly adopted in the content-based recom-

mender system (Pollack et al., 2003) (Pazzani et al., 1996), which can be employed to give a recommendation for user's activity.

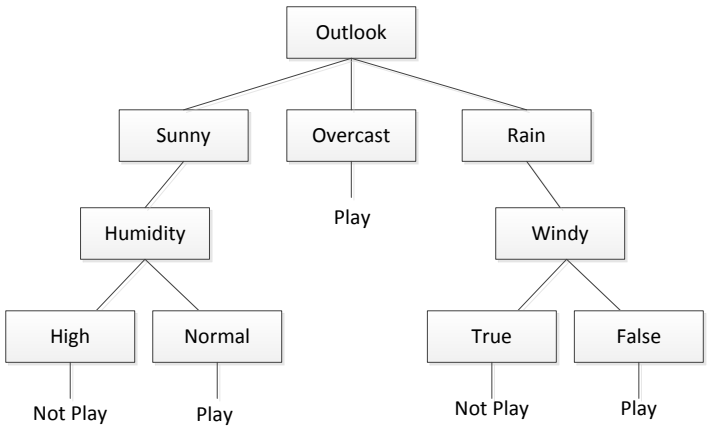


Figure 5.11. A Classic Example of a Decision Tree

5.3.1.2.3 Hidden Markov Model (HMM)

The HMM is the most accepted algorithm in temporal recognition tasks including speech, gesture, activity, etc. The HMM is a statistical Markov model, which can recover a data sequence that is not immediately observable. In human activity recognition, complex activities have a temporal structure. The time series data obtained by sensors is divided into time slices of constant length, where each slice is labeled with a state of activity. A generic HMM for activity is illustrated as shown in Figure 5.12.

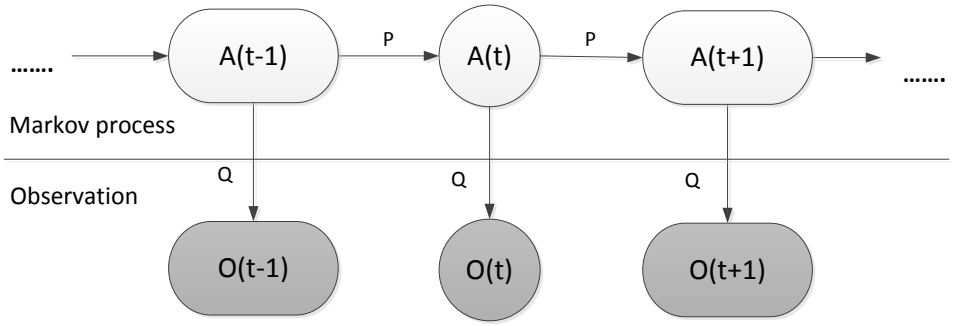


Figure.5.12 The Simple Example of HMM for Activity Recognition

The shaded nodes represent observable variables (data from sensors), while the white nodes represent hidden ones (state of activity). P is the state transition probabilities, Q is observation probability.

The Hidden Markov Model (HMM) is a generative probabilistic model consisting of a hidden variable and an observable variable at each time step (Baum et al., 1970). In activity context recognition, the hidden variable is a slice of activity, and the observable variable is extracted by sensors. There are two dependency assumptions for this model, represented with the directed arrows in Figure 5.12.

- The observable variable at time t, namely $\mathrm{O}(t)$, depends only on the hidden variable y_t at that time slice.

- The hidden variable at time t, namely $A(t)$, depends only on the previous hidden variable $\mathrm{O}(t)$.

To explain this clearly, we provide notations below

T observation times: $t = 0,....,\mathrm{T}-1$

N states: $Q_0,....,Q_{N-1}$

M observation symbols: $v_0,....,v_{M-1}$

State transition probabilities: $a_{ij} = \Pr(\mathrm{Q}_j \text{ at time t+1} \mid \mathrm{Q}_i \text{ at time t})$

Symbol probabilities: $b_j(\mathrm{k}) = \Pr(\mathrm{v}_j \text{ at time t} \mid \mathrm{Q}_j \text{ at time t})$

Initial state probabilities: $\pi_i = \Pr(\mathrm{Q}_i \text{ at time t})$

Observation sequence: $O = o_0, o_1,...., o_{t-1}$

State sequence: $A = a_0, a_1...., a_{t-1}$

One of the most important and difficult problems in HMMs is to estimate model parameters $\lambda = \{A, B, \pi\}$ from data. HMMs are trained with a Maximum Likelihood criterion: seek model parameters λ that best explain the observations, as measured by $\Pr(O \mid \lambda)$. This problem is solved with an iterative procedure known as Baum-Welch (Baum et al., 1970), which is an implementation of the EM algorithm. The main idea is to estimate π_i, a_{ij} and $b_j(k)$ by formulas as follows:

$$\pi_i = \frac{Expected\ number\ of\ q_1 = \mathrm{i}}{Number\ of\ training\ sequences}$$

$$a_{ij} = \frac{Expected\ number\ of\ transitions\ \ from\ state\ i\ to\ state\ \ j}{Expected\ number\ of\ transitions\ \ from\ state\ i}$$

$$b_j(o_k) = \frac{Expected\ number\ of\ "k"\ in\ state\ j}{Expected\ number\ of\ times\ in\ state\ j}$$

The Baum-Welch algorithm is one of the most popular methods in leaning phrase for HMM, and we adopt it in the HMM reasoner. The pseudo code is provided to show the basic process of the Baum-Welch algorithm in Figure 5.13.

Algorithm Baum-Welch (an implementation of EM algorithm)

Inputs: T, observations of length
V, output vocabulary
Q, hidden state set

Output: *HMM = (A, B)*

initialize A and B

iterate until convergence

E-Step

$$\gamma_t(j) = \frac{\alpha_t(j)\beta(j)}{P(O \mid \lambda)} \quad \forall t \text{ and } j$$

$$\xi_t(i,j) = \frac{\alpha_t(i)\, a_{i,j}\, b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(N)} \quad \forall t, i \text{ and } j$$

M-Step

$$\overline{a_{ij}} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1}\sum_{j=1}^{N} \xi_t(i,j)}$$

$$\overline{b_{ij}}(o_k) = \frac{\sum_{t=1, O_t = o_k}^{T-1} \gamma_t(j)}{\sum_{t=1}^{T-1} \gamma_t(j)}$$

return A, B

Figure.5.13 The pseudo code of the Baum-Welch algorithm

The learned HMM model can be used to recognize the dynamic activity. It means that inference which best labeled sequence explaining the new coming data from the sensors. It depends on calculating a maximum of the conditional probability:

$$a_1, a_2, a_3 ... = \underset{all\ a_1, a_2, a_3 ...}{ArgMax}\, P(a_1, a_2, a_3 ... \mid o_1, o_2, o_3 ...)$$

where a_i presents a state of activity, and o_i refers to observable variables from the sensors. The Viterbi algorithm is preformed efficient in the inference phrase, which was proposed by Andrew Viterbi in 1967 as a decoding

algorithm for convolutional codes over noisy digital communication links (Viterbi, 1967). The Viterbi algorithm can be simply described as an algorithm which finds the most likely path through a trellis, i.e. the shortest path, given a set of observations.

It contains four main steps:

1. Initialization ($t = 0$) :

$$\delta_0(\mathrm{i}) = \pi_i b_i(\mathrm{O}_0),\ 0 \le i \le N-1$$
$$\psi_1(\mathrm{i}) = 0$$

2. Time Recursion:

$$For\ 1 \le t \le T-1,\ 0 \le j \le N-1$$
$$\delta_t(\mathrm{j}) = \max[\delta_{t-1}(\mathrm{i})\,\mathrm{a}_{ij}]\, b_i(\mathrm{O}_0)$$
$$\psi_t(\mathrm{j}) = \arg\max[\delta_{t-1}(\mathrm{i})\,\mathrm{a}_{ij}]$$

3. Termination:

$$P_{\max} = \max[\delta_{T-1}(\mathrm{i})]$$
$$i_{T-1} = \arg\max[\delta_{T-1}(\mathrm{i})]$$

4. State sequence backtracking:

$$For\ t = T-2, T-3, \ldots 0$$
$$i_t = \psi_{t+1}(\mathrm{i}_{t+1})$$

The Viterbi algorithm is an efficient approach for finding the most probable state path through the model generating sequences. We provide the pseudo code to show the implementation process of the Viterbi algorithm in Figure 5.14.

Algorithm Viterbi

Inputs: T, observations of length
N, state-graph of length

Output: *best-path*

Create a path probability matrix viterbi[N+2,T]

for each state s **from** 1 **to** N **do** — **initialization step**

$viterbi\,[s,1] \leftarrow a_{0,s} \times b_s(o_1)$

$backpointer\,[s,1] \leftarrow 0$

for each time step t **from** *2* **to** T do — **recursion step**

for each state s **from** 1 **to** T do

$viterbi\,[s,1] \leftarrow \max_{s'=1} viterbi[s', t-1] * a_{s',s} * b_s(o_t)$

$backpointer\,[s,1] \leftarrow \max_{s'=1} viterbi[s', t-1] * a_{s',s}$

$viterbi\,[q_F,T] \leftarrow \max_{s=1} viterbi[s,T] * a_{s,q_F}$ — **termination step**

$backpoint\,[q_F,T] \leftarrow \max_{s=1} viterbi[s,T] * as_{,q_F}$

return the backtrace path by following backpointers to
states back in time from $backpoint\,[q_F,T]$

Figure.5.14 The pseudo code of the Viterbi algorithm

The HMM has become a popular tool in many recognition tasks, due to its powerful representation and tractability. As the focus of this dissertation is not the study of HMM but its utilization, we merely provide a simple explanation of HMM in mathematics: detailed information can be found in (Quinlan, 1986). In our intelligent inference engine, we adopt the HMM reasoner to take charge of dynamic activity analysis.

5.3.2 Organization of Three Algorithms

Our context-aware middleware enables application designers to concentrate on the development of application logic. The intelligent inference engine takes into account two aspects of design: expandability and scalability. Rules, DT, and HMM play important roles in basic activity inference, future activity recommendation, and dynamic activity analysis, respectively. The three algorithms are related to various implementation methods. For example, besides ID3 (Iterative Dichotomiser 3) (Quinlan, 1986) that we adopt, there are many specific decision-tree algorithms. Notable ones include: C4.5 (successor of ID3) (Quinlan, 1993), CART (Classification and Regression Tree) (Breiman, 1984), CHAID (CHi-squared Automatic Interaction Detector)(Kass, 1980). Different algorithms have different advantages to meet requirements. A mechanism is required to support expandability and scalability for the intelligent inference engine.

We adopt the strategy pattern (Freeman et al., 2004) to organize the three algorithms. This encapsulates the algorithm into separate classes, which enable the context-aware application developer to vary the algorithm independently from the context and to plug in a new one at runtime. The strategy pattern offers an alternative to conditional statements for selecting desired behavior, which makes the three algorithms interchangeable. It gives this module flexibility, so that context-aware application developers can alter and extend the module, such as adding other algorithms without affecting the originals, as long as the algorithms employ the correct interface.

We define a universal interface "InferBehavior" and a class "InferenceEngine", which contains three subclasses: "BasicInference", "RecommendationInference", and "DynamicAnalysis" supported by Rules, the Decision Tree and the Hidden Markov Model. If context-aware application developers want to adopt different inference methods to deal with different situations, they just invoke the function with the same name based on this interface, and do not need to use the actual algorithm. The partial UML model is shown in Figure 5.15.

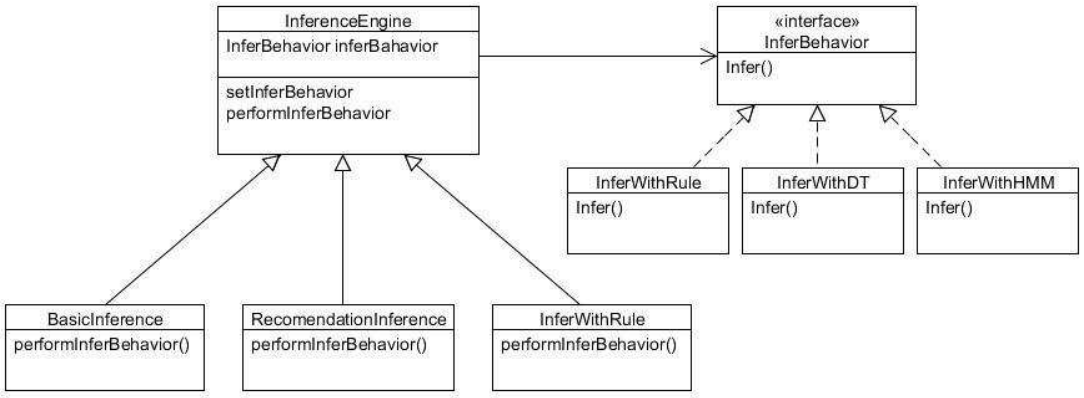


Figure.5.15 The Partial UML of the Inference Engine

5.3.3 The Invoking Mechanism

The invoking mechanism refers to how and when the context-aware middleware runs the inference engine. According to different tasks of reasoners in the intelligent inference engine, we present different methods for the reasoners.

In the basic module, the OWL reasoner's main task is to check context consistency. It is invoked when the new context model is added to, such as adding a new specific model. In the intelligent module, due to the different methods of activity context recognition, the invoking mechanism is more complex than the basic module.

Concerning the Rules reasoner, we design a query invoking mechanism that enables the rules reasoner to work when receiving the query from an external service, such as context-aware applications. There is a very interesting phenomenon, i.e. the property exists in almost all the query statements. For example: "?s ex:hasLocated ex:Room3", furthermore, the property is like a linking point of the objects, the amount of which is much less than the object itself. So these properties are defined as the keywords, which are used to trigger the specific users' rules set in the Rules reasoner. When a query statement arrives, the context-aware middleware reacts: it parses the query statement, then tries to match keywords. If it succeeds, the basic inference engine is invoked, while if it fails, it searches for the context database directly. The entire flow is shown in Figure 5.16.

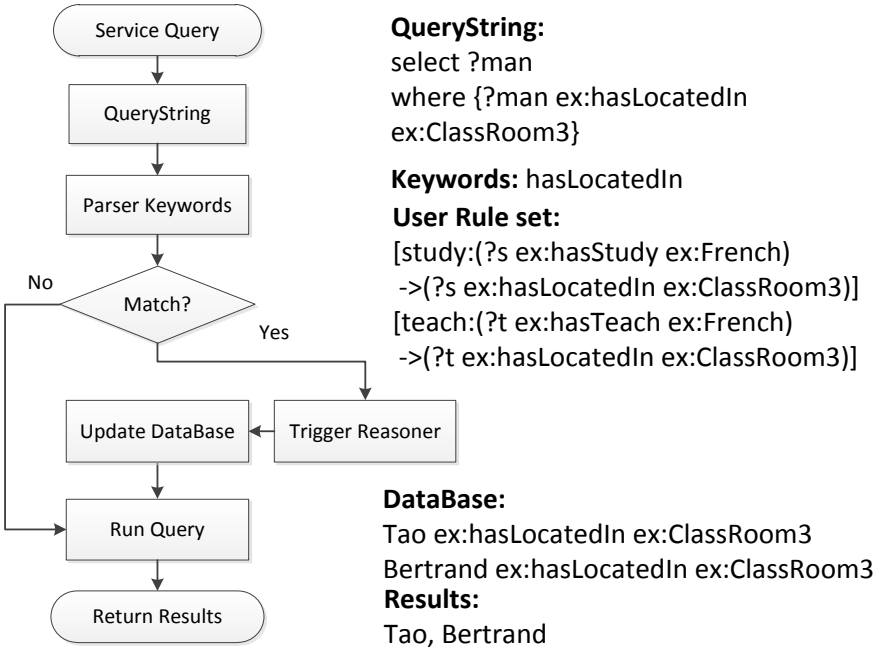


Figure.5.16 The flow chart of the query invoking mechanism

The Rules reasoner works as the basic inference used most frequently. Context-awareness application developers formulate the different rule sets for different activities. Based on efficiency considerations, the Rules reasoner

will work on the different rule sets according to queries relating to types of activities.

With respect to the DT reasoner, we present a schedule invoking mechanism, which invokes the DT reasoner based on the user's schedule. The DT reasoner is in charge of future activity recognition, which works with the user's schedule. When the system finds that the user is not in the place where the user is scheduled, the system reminds the user and gives a suggested choice based on the user's current location. In the bus stop scenario, we focus on providing the user with an appropriate restaurant located nearby. Figure 5.17 explains the flow of schedule of the trigger mechanism.

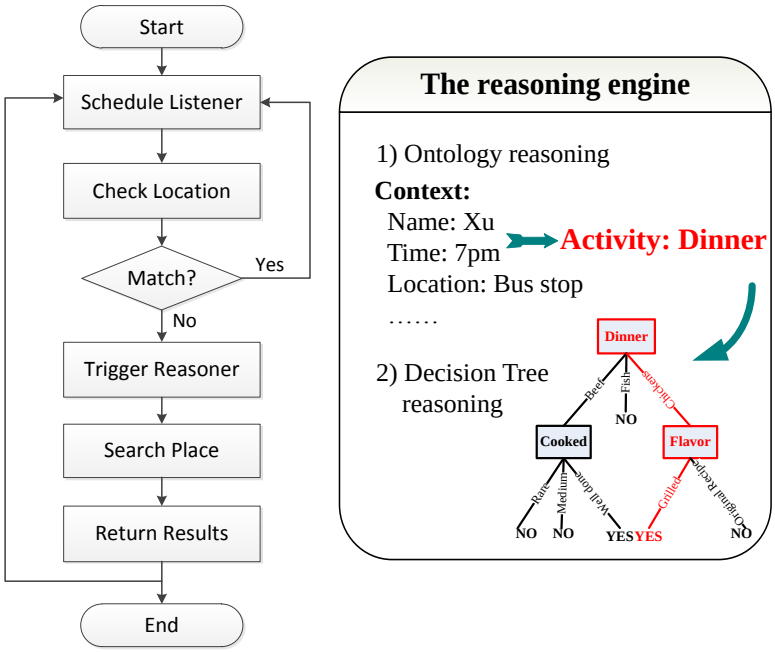


Figure.5.17 The flow chart of the schedule invoking mechanism

For the HMM reasoner, we enable it to work always when the middleware system is operating, due to the requirements of the reasoner's algorithm and task. As it provides the activity sequence inference based on sequence sensor data, it requires time series data from sensors. The inference results and the historical results can be stored in the context knowledge base. The context-aware application can employ the results directly based on different requirements.

5.4 Conclusion

In this chapter, since research work revolves increasingly around the "activity" context instead of the "location" context, we reviewed literatures on activity context recognition in three premier conferences in context awareness over the past ten years, summarized all the methods, and divided research on activity context recognition into three main facets: basic activity inference, dynamic activity analysis, and future activity recommendation.

Based on our previous work, we proposed an intelligent inference engine for context-aware middleware, consisting of a basic inference module and an intelligent inference module. Besides satisfying requirements for checking context consistency, our inference engine integrates the three most popular methods for activity context recognition: Rules, the Decision Tree, and the Hidden Markov Model. It provides a solution for all facets of activity context recognition based on our context-aware middleware. However, intelligent inference did not take into consideration the context ambiguity problem, i.e. that contexts from the sensors are not always correct. The context inference engine will reach the conclusion based on inaccurate information, which will lead to taking incorrect actions of application, especially in the Rules reasoner. We will improve context inference in this field to make our system more robust and intelligent.

The content of this chapter was published in:

Xu, T., Zhou, Y., David, B., Chalon, R., Supporting Activity Context Recognition in Context-aware Middleware, Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI'13), Bellevue, Washington, USA.

6 Implementations and Analysis

6.1 Introduction

Although some solutions already exist, developing a context-aware application is still a tough task. This chapter explains how to use our context-aware middleware to build a context-aware application: how to build the context representation model, how the context-aware middleware works, as well as how the intelligent inference engine deals with three facets of problems concerning user activity recognition via two scenarios (applications).

First, it provides an instruction of the general process to build a context-aware application in ambient intelligence. Then, the bus stop scenario will be presented for interpreting context modeling and different designers' contributions to the context model, and for describing each component's tasks in context-aware middleware in which the intelligent inference engine is related to the Rules reasoner and the DT reasoner. Next, the domestic activity application is used to verify the HMM reasoner. Finally, we compare our middleware with five representative context-aware systems and conclude as to its advantages and disadvantages.

6.2 General process for building a context-aware application

This section will illustrate the general process for developing a context-aware application by our context-aware middleware. It is related to three steps: context modeling, middleware configuration, and application development.

6.2.1 Context Modeling

As different context-aware applications work for different situations, different types of context are required. In our context representation model, the generic model uses four interrelated base classes: user, location, time, and activity, representing who, where, when and what for activities of entities. The device class is used to organize different devices used in the environment. This is just a general description and can be extended by more precise contextual information via specific models. There are two general steps to define the user's context model to meet different requirements:

1. Figure out what types of context are required by the application.
In ubiquitous computing, the context used in health-care applications is very different from those used in mobile learning applications.

2. Extend the context representation context by new required context.
The spatial-temporal model provides a large amount of type context but this is not sufficient to model all information used in applications. Thanks to adoption of ontology technology, application developers can add new ontology classes and properties directly as required. Furthermore, the existing suitable context model can be imported to the users' context model as well.

6.2.2 Middleware Configuration

Our context-aware middleware provides a solution for developing context-aware applications. Application developers can use it directly or modify some functions according to application requirements. Two main components (context aggregator and context intelligent inference engine) in context-aware middleware are scalable and extensible. They can be configured before developing applications.

1. Context aggregator configuration

The context aggregator's main task is to convert raw information extracted from the low layer to information described in OWL format. How it works depends on the corresponding context resource, such as physical sensors. The middleware provides some basic context aggregator components, such as the location context aggregator component, and defines a universal interface for extension. Before developing applications, developers can add new aggregator components to meet new requirements. Following this universal interface, any new user-defined aggregator components will work well with other components in context-aware middleware.

2. Context inference engine configuration

The context inference engine is an intelligent component in our context-aware middleware. One of its most important tasks is to deduce user's activity from some low-level context. It integrates three reasoners: Rules reasoner, DT reasoner and HMMs reasoner to deal with three facets of problems concerning user activity recognition. The three reasoners can satisfy the basic requirements. Resembling the context interpreter, context inference also defines a universal interface for extension. Application developers can modify the algorithm in the reasoner according to requirements, and can even add a new reasoner following the universal interface.

6.2.3 Application Development

Our context-aware middleware can hide the complex details of context retrieving and context processing, helping developers focus on designing the application logic. Context-aware middleware provides basic functions for handling context inference, expressive query and context storage. Besides considering the application function, developers should set context inference and context query.

1. Training and setting rules for Reasoners

As context-aware applications increasingly focusing on user's activity are developed, our middleware improves system intelligence by supporting activity context recognition. The three reasoners in the intelligent context inference engine are responsible for this task. The Rules reasoner uses first-order logic to deduce user's activities by rule sets, so developers need to

define rule sets for the relevant activity in advance. The DT reasoner and HMM reasoner have adopted artificial intelligence algorithms. To function properly, they need training data (user's previous activity) to calculate the parameters. If the application needs this kind of high-level context, the developer should train related reasoners first. Collecting users' previous activity data is not an easy task: we provide a solution that extracts training data on a specific activity from the user's social networks. The process will be interpreted in the bus stop scenario.

2. Define expressive query

Since the knowledge base is used to store contexts, it differs from the conventional database, requiring utilization of semantics query language (SPARQL) to handle the expressive query. It requires application developers to specify context queries in the form of SPARQL. Furthermore, another function of the query engine is to invoke the Rule reasoner. Application developers should take into account the relationship between query and rule set in advance.

6.3 Scenario and Implementation

In this section, we propose two scenarios (applications) to illustrate the development process. For the intelligent inference engine, the bus stop scenario involves the Rules reasoner and the DT reasoner, while the domestic activity application is used to verify the HMM reasoner.

6.3.1 Bus Stop Scenario

The bus stop application is a typical application in the Smart City. In our previous work, we organized all activities around or in relation to the bus stop. The bus stop can provide hot spot services and location-based services. To explain more clearly how the context-aware middleware works, we present the bus stop scenario as follows: after an international conference, Tao is taking the bus back to his hotel. He is tired, hungry and only wishes to have his favorite meal: roast chicken. However, he has never been to this city before and knows nothing about it. While he is fantasizing about this food, the bus arrives at the bus stop. "Oh-la-la!" Tao gets off the bus and shouts out with excitement. An avatar, recognizing him (by the collection of his identification data) in the large public screen speaks his native language to him with the subtitles popping up: "Welcome! The roast chicken

restaurant is about 300m away from this bus stop. If you want to book a seat, please wave your hand to me…

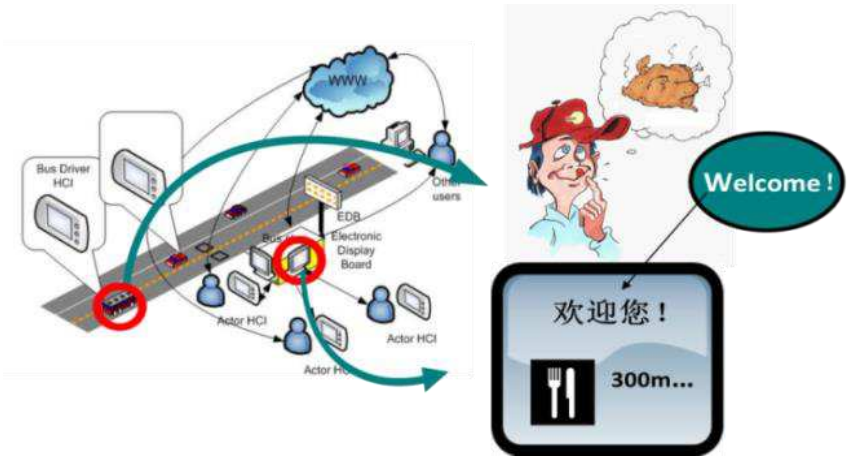
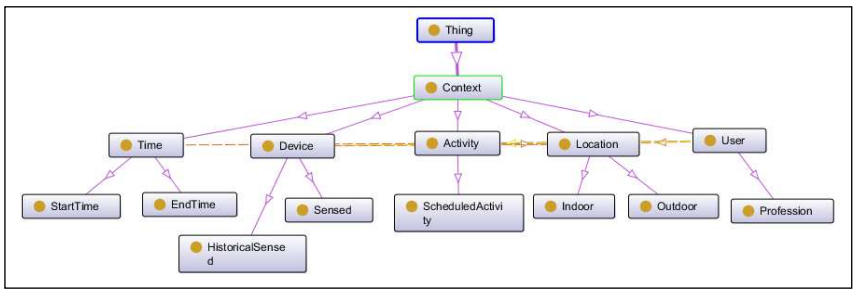


Figure.6.1 The bus stop scenario

6.3.1.1 Implementation Process

6.3.1.1.1 Step one: Context Modeling

As described earlier, context modeling is the first step in developing context-aware applications. The generic model pre-defines the fundamental ontology classes: location, activity, user, time and device, and the main properties between ontology classes. Figure 6.2 (a) shows the architecture of the generic model, developed in protégé; the partial basic ontology class with OWL format is shown in Figure 6.2 (b), while the partial main properties with OWL format is shown in Figure 6.2 (c). The generic model is developed by context-aware middleware designers. The application cannot directly modify it but can extend it by adding specific models or importing some existing context models.



(a) The generic model

```
<!--
///////////////////////////////////////////////////////////////////////////////////////
//
// Classes
//
///////////////////////////////////////////////////////////////////////////////////////
 -->

<!-- http://liris.cnrs.fr/smartspace.owl/BusStop.owl#Activity -->

<owl:Class rdf:about="http://liris.cnrs.fr/smartspace.owl/BusStop.owl#Activity">
    <rdfs:subClassOf rdf:resource="http://liris.cnrs.fr/smartspace.owl/BusStop.owl#Context"/>
</owl:Class>



<!-- http://liris.cnrs.fr/smartspace.owl/BusStop.owl#Context -->

<owl:Class rdf:about="http://liris.cnrs.fr/smartspace.owl/BusStop.owl#Context"/>



<!-- http://liris.cnrs.fr/smartspace.owl/BusStop.owl#Device -->

<owl:Class rdf:about="http://liris.cnrs.fr/smartspace.owl/BusStop.owl#Device">
    <rdfs:subClassOf rdf:resource="http://liris.cnrs.fr/smartspace.owl/BusStop.owl#Context"/>
</owl:Class>



<!-- http://liris.cnrs.fr/smartspace.owl/BusStop.owl#EndTime -->

<owl:Class rdf:about="http://liris.cnrs.fr/smartspace.owl/BusStop.owl#EndTime">
    <rdfs:subClassOf rdf:resource="http://liris.cnrs.fr/smartspace.owl/BusStop.owl#Time"/>
</owl:Class>



<!-- http://liris.cnrs.fr/smartspace.owl/BusStop.owl#HistoricalSensed -->

<owl:Class rdf:about="http://liris.cnrs.fr/smartspace.owl/BusStop.owl#HistoricalSensed">
    <rdfs:subClassOf rdf:resource="http://liris.cnrs.fr/smartspace.owl/BusStop.owl#Device"/>
</owl:Class>
```

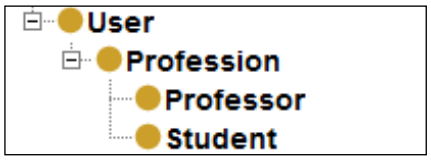
(b) The partial classes in generic with OWL

```
<!--
///////////////////////////////////////////////////////////////////////////////////////
//
// Object Properties
//
///////////////////////////////////////////////////////////////////////////////////////
 -->


<!-- http://liris.cnrs.fr/smartspace.owl/BusStop.owl#At -->

<owl:ObjectProperty rdf:about="http://liris.cnrs.fr/smartspace.owl/BusStop.owl#At">
    <rdfs:domain rdf:resource="http://liris.cnrs.fr/smartspace.owl/BusStop.owl#Activity"/>
    <rdfs:range rdf:resource="http://liris.cnrs.fr/smartspace.owl/BusStop.owl#Time"/>
</owl:ObjectProperty>



<!-- http://liris.cnrs.fr/smartspace.owl/BusStop.owl#EngagedIn -->

<owl:ObjectProperty rdf:about="http://liris.cnrs.fr/smartspace.owl/BusStop.owl#EngagedIn">
    <rdfs:range rdf:resource="http://liris.cnrs.fr/smartspace.owl/BusStop.owl#Activity"/>
    <rdfs:domain rdf:resource="http://liris.cnrs.fr/smartspace.owl/BusStop.owl#User"/>
</owl:ObjectProperty>



<!-- http://liris.cnrs.fr/smartspace.owl/BusStop.owl#LocatedIn -->

<owl:ObjectProperty rdf:about="http://liris.cnrs.fr/smartspace.owl/BusStop.owl#LocatedIn">
    <rdfs:domain rdf:resource="http://liris.cnrs.fr/smartspace.owl/BusStop.owl#Activity"/>
    <rdfs:domain rdf:resource="http://liris.cnrs.fr/smartspace.owl/BusStop.owl#Device"/>
    <rdfs:range rdf:resource="http://liris.cnrs.fr/smartspace.owl/BusStop.owl#Location"/>
    <rdfs:domain rdf:resource="http://liris.cnrs.fr/smartspace.owl/BusStop.owl#User"/>
</owl:ObjectProperty>
```

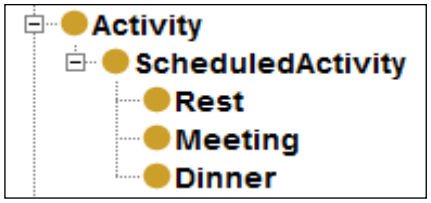
(c) The partial object properties in generic with OWL

Figure.6.2 Our context model

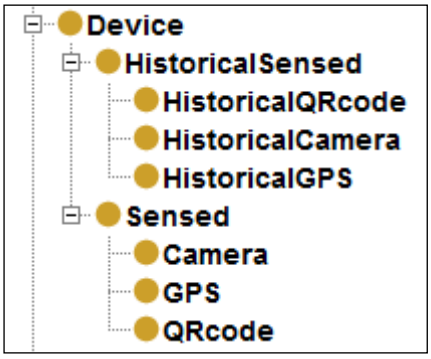
For the bus stop scenario, some subclass is added as the specific model to complete the generic class. As the purpose of the study is the development process, this specific model is just built for activities around conference buildings, bus stops and restaurants. If necessary, the other ontology-based specific models can be added easily. As mentioned earlier, the specific model is developed by context-aware application designers. They can freely define new subclasses to refine the generic model. Our partial specific model is shown in Figure 6.3. This scenario is related to two types of users: professor and student, shown in Figure 6.3 (a); Figure 6.3 (b) shows some related activities; the partial devices used in this scenario are shown in Figure 6.3 (c); the partial location place involved in the scenario is presented in Figure 6.3 (d).



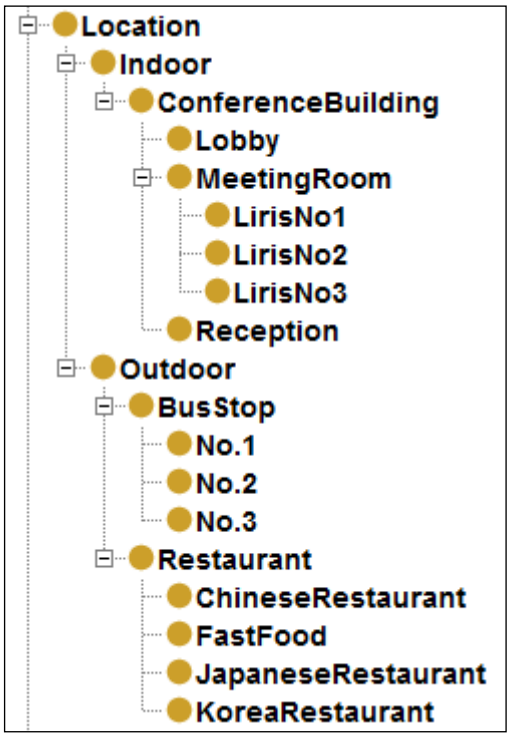
(a) Specific subclass for User



(b) Specific subclass for Activity



(c) Specific subclass for Device



(d) Specific subclass for Location

Figure.6.3 The bus stop scenario

The individuals (instances) come from users or context-aware environments. They can be pre-defined or obtained from the sensors or other context resources. There are some examples of important individuals used in the bus stop scenario, shown in Figure 6.4. Figure 6.4 (a) shows an individual of the "Student" class: Tao, which provides the user's profile via "datatype" such as male, home address. An individual of the "Chinese restaurant" class is shown in Figure 6.4 (b). It contains the detailed information of the restaurant that can be used in the inference engine, such as flavor, service and price.

```
<!-- http://liris.cnrs.fr/smartspace.owl/BusStop.owl#Tao -->

<owl:NamedIndividual rdf:about="http://liris.cnrs.fr/smartspace.owl/
      BusStop.owl#Tao">
    <rdf:type rdf:resource="http://liris.cnrs.fr/smartspace.owl/
      BusStop.owl#Student"/>
    <PhoneNumber rdf:datatype="&xsd;string">0678675800</PhoneNumber>
    <HomeAdress rdf:datatype="&xsd;string">51 chemine des Mouilles
    </HomeAdress>
    <Sex rdf:datatype="&xsd;string">male</Sex>
</owl:NamedIndividual>
```

(a) The individual for "Tao"

```
<!-- http://liris.cnrs.fr/smartspace.owl/BusStop.owl#Hotpot -->

<owl:NamedIndividual rdf:about="http://liris.cnrs.fr/smartspace.owl/
      BusStop.owl#Hotpot">
    <rdf:type rdf:resource="http://liris.cnrs.fr/smartspace.owl
    /BusStop.owl#ChineseRestaurant"/>
    <Environment rdf:datatype="&xsd;int">19</Environment>
    <Flavor rdf:datatype="&xsd;int">23</Flavor>
    <Service rdf:datatype="&xsd;int">29</Service>
    <Level rdf:datatype="&xsd;float">4.5</Level>
    <Price rdf:datatype="&xsd;int">57</Price>
    <Address rdf:datatype="&xsd;string">xxx street</Address>
</owl:NamedIndividual>
```

(b) The individual "hotpot"

Figure.6.4 The individual examples

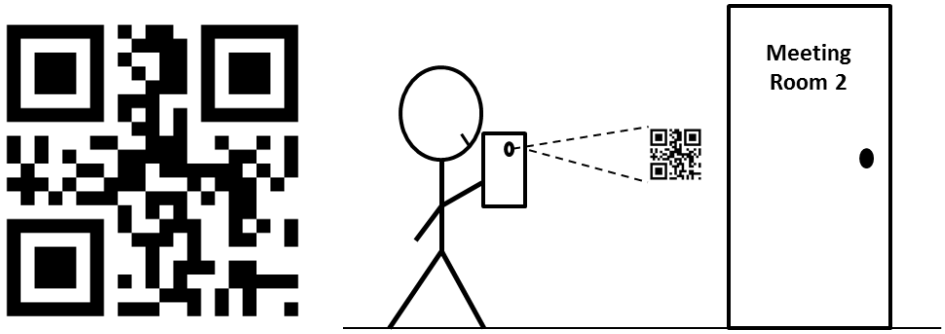
6.3.1.1.2 Step Two: Middleware Configuration

As described earlier, middleware configuration involves two main processes: context aggregator configuration, context inference engine configuration.

6.2.1.1.2.1 Context Aggregator Configuration

Location context is one of the most important contexts in ubiquitous computing. We will take the location context aggregator as an example to interpret context aggregator configuration.

In the bus stop scenario, the main purpose of the study is to test our context-aware middleware. GPS devices work for the location information outside. For location in the building, we design a very simple indoor location system by using QRcode and camera to test middleware. The basic idea is that the room information is encoded into QRcode, and users can scan the code to locate themselves in the building by wearable camera or camera fixed in their mobile phone. A simple example is shown in Figure 6.5.



(a) "Meeting Room 2"in QRcode (b) the basic idea

Figure.6.5 The indoor location system

One of the main functions of the indoor location system is shown in Figure 6.6 (a), which is responsible for decoding the QRcode and returning the results in string format via ZXing and JavaCV. It is published as a web service, which can be invoked by different functions directly. The partial wsdl file is presented in Figure 6.6 (b). The raw location context that is gotten from the ubiquitous environments cannot be used directly by the application. To be used by other components in context-aware middleware, it will be sent to be processed by the location context aggregator first.

```java
public String qrcodeReader() throws Exception {
    // initialize seting
    Result result = null;
    BinaryBitmap bitmap;
    OpenCVFrameGrabber grabber = new OpenCVFrameGrabber(0);
    CanvasFrame canvasFrame = new CanvasFrame("Original stream");
    grabber.start();
    IplImage frame;
    boolean isRunning = true;
    // Read QRcode
    while (isRunning) {
        if ((frame = grabber.grab()) == null)
            break;
        canvasFrame.setCanvasSize(frame.width(), frame.height());
        canvasFrame.showImage(frame);
        // get QRcode image
        LuminanceSource source = new BufferedImageLuminanceSource(
                frame.getBufferedImage());
        // load QRCode
        bitmap = new BinaryBitmap(new HybridBinarizer(source));

        try {
            // decode QRcode
            result = new MultiFormatReader().decode(bitmap);
            isRunning = false;
        } catch (NotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } }

    grabber.stop();
    canvasFrame.dispose();
    // Return information
    return result.getText();
}
```

(a) One main function of the indoor location system

```xml
<wsdl:portType name="QRCodeReader">
   <wsdl:operation name="readImage">
      <wsdl:input message="impl:readImageRequest" name="readImageRequest">
    </wsdl:input>
      <wsdl:output message="impl:readImageResponse" name="readImageResponse">
    </wsdl:output>
   </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="QRCodeReaderSoapBinding" type="impl:QRCodeReader">
   <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation name="readImage">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="readImageRequest">
         <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="readImageResponse">
         <wsdlsoap:body use="literal"/>
      </wsdl:output>
   </wsdl:operation>
</wsdl:binding>
```

(b) The partial wsdl file for the "qrcodeReader" function

Figure.6.6 The sample code of the indoor location system

The context aggregator's main task is to convert raw contextual information gotten from the low layer to context described in OWL format and update to the context knowledge base. Context is stored in the context knowledge base with a triple pattern, representing a relationship between entities. In the bus stop scenario, the location context aggregator is designed to combine the device holder and his location information gotten from the indoor location system to form a triple pattern and upload to the knowledge base. Also, application developers can define their own location context aggregator by implementing the interface: "ContextAggregator". The sample code of our location context aggregator is shown in Figure 6.7 (a), which contains one of the main functions in the "LocationContextAggregator" class for the bus stop scenario. The results generated in OWL format are shown in Figure 6.7 (b)

```
public class LocationContextAggregator implements ContextAggregator{

   final static String ex = "http://liris.cnrs.fr/smartspace.owl#";
   @Override
   public Statement convertContextToStatement
               (String person, String location, OntModel m) {

       // create a temp model to void trigger "add statement to model" event
       OntModel tempModel = ModelFactory.createOntologyModel();
       tempModel.add(m);

       // create the new person individual
       OntClass student =
           tempModel.getOntClass(ex+"Student");
       Individual subject =
           tempModel.createIndividual(ex+person, student);

       // get the Property
       ObjectProperty hasLocatedIn =
           tempModel.getObjectProperty(ex+"LocatedIn");

       //create the new location individual
       OntClass MeetingRoom =
           tempModel.getOntClass(ex+"MeetingRoom");
       Individual object =
           tempModel.createIndividual(ex+location, MeetingRoom);

       Statement locationStatement =
           tempModel.createStatement(subject, hasLocatedIn, object);

       tempModel.close();
       return locationStatement;
   }
```

(a) The sample function in the location context aggregator

```
<Student rdf: about = "Tao">
        <LocatedIn rdf:about = "Meeting Room 2">
</Student>
```

(b) The results generated in OWL format

Figure.6.7 The sample code for the location context aggergator

6.2.1.1.2.2 Context inference engine configuration

The context inference engine is the intelligent component in context-aware middleware, and can be configured and modified by application developers according to requirements. As described in chapter five, context intelligent inference consists of two parts with four reasoners, dealing with two main tasks: checking context consistency and deducing users' activities. For convenient configuration, we organize the reasoners by strategy pattern. Context-aware application developers can easily extend and replace the reasoner by implementing the universal interface.

In the bus stop scenario, this involves two activity reasoners: Rules reasoner and DT Reasoner. The Rules reasoner provides the basic inference for user's activities based on the user's current context, such as meeting, dining, taking the bus, and so on. The DT Reasoner recommends the user's future activities based on analysis of the user's previous activities. The Rules reasoner is developed based on Jena ("Apache Jena," 2010), which supports both forward chaining inference and backward chaining inference. The DT reasoner uses J48 implementation of the C4.5 decision tree in Weka (Hall et al., 2009), which is an open source on data mining in Java to provide a collection of machine learning algorithms. All the reasoners are implemented in the interface "InferenceBehavior". The sample code for one of the functions in the Rules reasoner is shown in Figure 6.8. If necessary, developers can modify or add new algorithms according to requirements.

```
public class InferWithRule implements InferBehavior {

    //define Reasoner
   GenericRuleReasoner reasoner;
   OntModel ontModel;
   Resource configuration;
   InfModel infModel;

   @Override
   public OntModel infer(String ruleFile, Model model) {

       //initial Reasoner
       ontModel = ModelFactory.createOntologyModel();
       List rules = Rule.rulesFromURL(ruleFile);
       reasoner = new GenericRuleReasoner(rules);
       reasoner.setOWLTranslation(true);
       reasoner.setTransitiveClosureCaching(true);

       //initial infModel
       infModel = ModelFactory.createInfModel(reasoner, model);
       //infer context
       infModel.prepare();
       //Load the results
       ontModel.add(infModel);
       //Return Results
       return ontModel;
   }
```

Figure.6.8 The sample code for the Rules reasoner

6.3.1.1.3 Step Three: Application Development

As described in the previous section, development of the context-aware application is different from conventional applications. Developers should take into account training the context inference and defining context query besides designing the application function.

6.2.1.1.3.1 Training and setting rules for Reasoners

Setting rules for Reasoners

The intelligent inference engine plays an important role in the bus stop scenario. To obtain the user's activity context, it uses two reasoners: Rules reasoner and DT reasoner. The Rules reasoner is responsible for inferring Tao's activity based on a rule set. Therefore, application designers should define the first-order logic rules for specific activities respectively, based on related low-level contexts that can be easily gotten from ubiquitous computing. To simplify understanding, Figure 6.9 lists some rules that we define for the bus stop scenario.

```
@prefix ex: <http://liris.cnrs.fr/smartspace.owl#>.

[Meeting:
  (?user rdf:type ex:User)^
  (?activity rdf:type ex:Meeting)^
  (?activity ex:LocatedIn ex:MeetingRoom2)^
  (?user ex:LocatedIn ex:MeetingRoom2)^
  (?activity ex:startTime ?t1)^
  (?activity ex:endTime ?t2)^
  greaterThan(?currentTime,?t1)^
              lessThan(?currentimeTime,?t2)

  ->(?user ex:EnagaedIn ?activity)]

[Discussing:
  (?user1 rdf:type ex:Student)^
  (?user2 rdf:type ex:Professor)^
  (?user1 ex:supervisorOf ?user)^
  (?user1 ex:LocatedIn ?Office)^
  (?user1 ex:LocatedIn ?Office)^
  (?user2 ex:hasOffice ?office)

  ->(?user1 ex:Discuss ?user2)]

[TakingBus:
  (?user rdf:type ex:User)^
  (?bus rdf:type ex:Bus)^
  (?user ex:stepOn ?bus)^
  noValue(?user ex:stepOff ?bus)

  ->(?user ex:takeBus ?bus)]

[AtSupper:
  (?user rdf:type ex:User)^
  (?resto rdf:type ex:Restaurant)^
  (?user ex:LocatedIn ?resto)^
  greaterThan(?currentTime,18:30:00)^
              lessThan(?currentimeTime,21:00:00)

  ->(?user ex:Dinning ?resto)]
```

Figure.6.9 The sample rules

Training Reasoner

The decision tree focuses on recommending the potential favorite restaurant to Tao. To achieve this recommendation, the three following steps are required: collecting the training data, building the decision tree, providing the predictive recommendation.

Collecting the training data is one of the foremost tasks for the DT reasoner. It requires collecting accurate and available information on the previous activities of every user. It has been considered a tough task in the past, since it is hard to let the user wear diverse sensors to travel around for a long time only to collect raw training data. The prevalence of social networks provides a possible solution to this. Increasingly people are posting their daily activities on their own social networks as part of life. This applies in particular to the microblog, whose content is typically smaller in both actual and aggregate file size. It is a convenient way for users to post their activity by mobile devices anywhere and anytime.

We chose the "weibo", as a data source to collect information on users' activities. (The "weibo", teeming with more than 300 million users, is the biggest twitter-like microblogging service in China). After analysis, we chose dinner activity as a research object. We collected available weibo microblogs by a keyword filter, which contains two keyword subsets: one refers to the specific restaurant's name: KFC, Mcdonalds, etc., while the other refers to the set of words usually appearing in the restaurant name: hot pot, restaurant, etc. An example in "weibo" is shown below. It contains the restaurant's name, address, user's preference and visiting time.

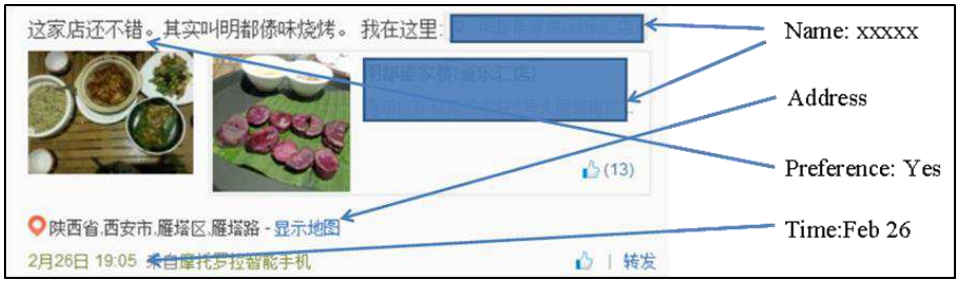


Figure.6.10 An example of User's Weibo

Detailed quantitative information is necessary for each restaurant, such as price, flavor, and environment. The site: www.dianping.com is opted for as the restaurant's detailed quantitative data source. This is an online independent third-party consumer service rating sites, which contains eight sorts of restaurant information: name, price, flavor, service, type, etc., as shown in Figure 6.11. For convenience, the type of restaurant is redefined based on nationality and fast food.

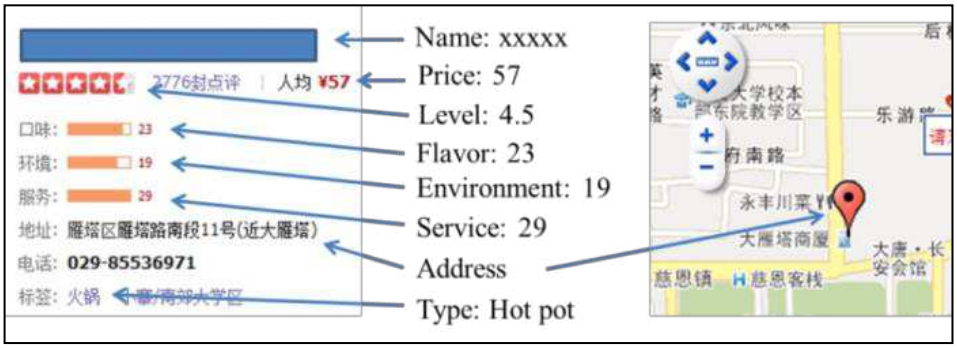


Figure.6.11 The Restaurant Information from www.dianping.com

The user's favorite restaurants from "weibo", along with their detailed quantitative information, construct the training set for choosing a restaurant.

In this scenario, a real weibo user's microblog information is used with his permission. We collect the data from May 12th, 2011 to May 2nd, 2012. The clawer has collected 14 available weibo's micro blogs about restaurants. These chosen restaurants construct a set of training data, as shown in Figure 6.12.

No.	Level	Price	Flavors	Environment	Service	Type	Choose
1	3.5	39	22	13	13	Chinese	yes
2	4	15	25	16	17	Chinese	no
3	4	27	19	18	19	Fast food	Yes
4	3	41	18	12	12	Chinese	No
5	5	60	25	25	31	Chinese	Yes
6	3.5	422	16	20	19	Chinese	No
7	5	56	25	23	24	Korea	Yes
8	3.5	55	18	17	15	Japan	No
9	3	24	16	17	15	Fast food	Yes
10	4.5	117	23	26	20	Japan	No
11	4	58	19	20	17	Fast food	Yes
12	4	183	17	24	20	Chinese	No
13	3	33	20	7	11	Chinese	Yes
14	3	7	20	9	10	Chinese	No

Figure.6.12 Training dataset

.

To build the decision tree, we use J48 implementation of the C4.5 decision tree in Weka (Hall et al., 2009), which is an open source on data mining in Java providing a collection of machine learning algorithms. The learned decision tree provides a profitable suggestion to help the user make a decision based on the training dataset. As shown in Figure 6.13, this tree is built to recommend restaurants for the chosen user.

The DT reasoner can be invoked based on Tao's schedule. In this scenario, when our context-aware middleware finds that Tao is not in the place where he is scheduled (he is not in a restaurant), it will remind Tao and suggest a choice (a favorite restaurant) based on his current location. The learned decision tree is used to select the favorite restaurant from the list of restaurants located nearby.

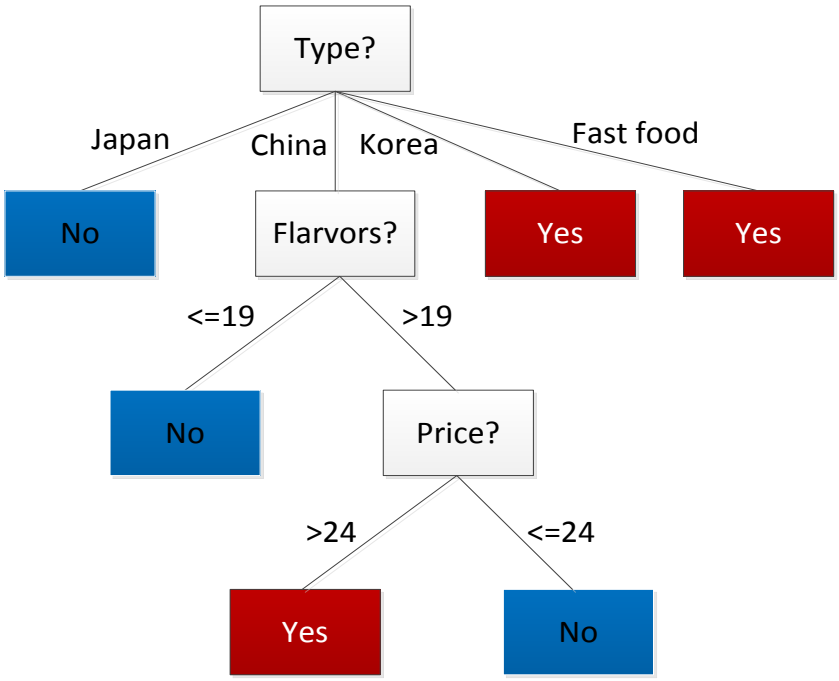
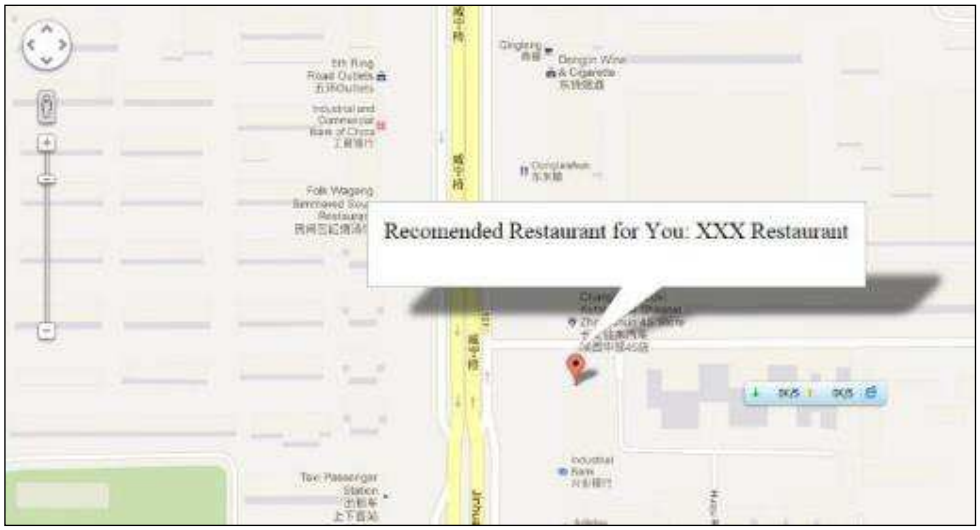


Figure.6.13 The Learned Decision Tree for Choosing Restaurants

The detailed information of the recommended restaurant is written into an xml file shown in Figure 6.14(a), and used in the Google map. In this way, the user gets a restaurant recommendation on the Google map, which helps him find this restaurant, as shown in Figure 6.14(b). Various devices, like smart phone, pad etc., can directly access the recommended restaurant on the map via internet.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <Restaurant>
    <Name>XXX Restaurant</Name>
    <Lon>34.235219</Lon>
    <Lat>108.933881</Lat>
    <Lev>8</Lev>
  </Restaurant>
```

(a) Detailed information in xml



(b) Recommended results in Google map

Figure.6.14 The Results for the DT Reasoner

6.3.1.1.3.2 Define expressive query

Define expressive query involves two main tasks: one is to query the context required by the context-aware application, and the other to invoke the Rules reasoner. As it is different from the conventional querying data, the querying context focuses on searching for sets of triples which describe situations of entities. The SPARQL query language adopted in our middleware provides competent support. Application developers can query any triples via the following SPARQL grammar rules. The latter task is the important point in defining expressive query. We have designed a query invoking mechanism that enables the rules reasoner to work when receiving the query from context-aware applications. The main idea is to parse the property in query statement and then invoke the corresponding rule set

based on this "property". The detailed information about the query invoking mechanism is described in chapter 5.5.3. We provide a simple example of this scenario in Figure 6.15. Based on this idea, context-aware application developers can freely define their queries and their invoked rule sets according to their application or environment requirements.

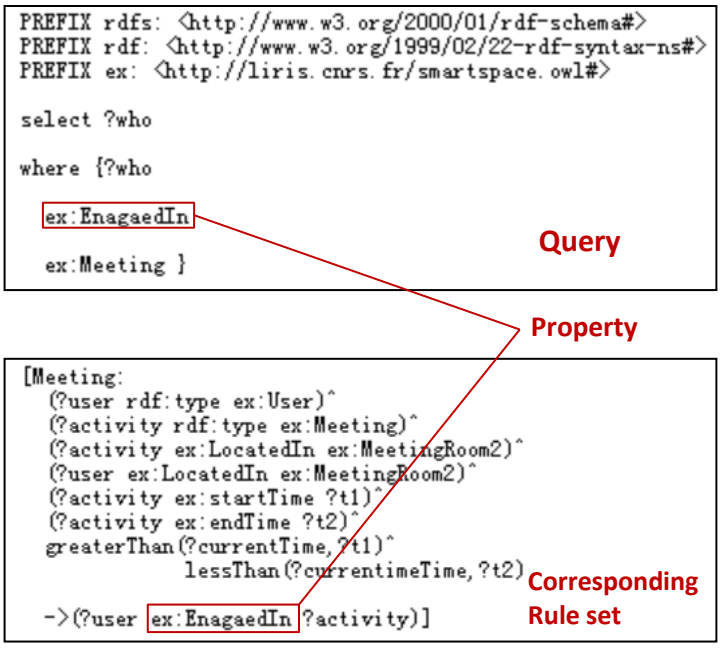


Figure.6.15 The sample for query and corresponding rule set

6.3.2 Domestic Activity Application

As a supplement, this subsection will focus on explaining how the HMM reasoner trains and works, and the detail of other steps will not be repeated. We use the domestic activity dataset from (van Kasteren et al., 2008) to verify the HMM reasoner. They employed 14 binary input sensors to record a user's seven kinds of daily activities: leaving the house, using the toilet, taking a shower, going to bed, preparing breakfast, preparing dinner and having drinks in an apartment as shown in Figure.6.16 from 25 Feb 2008 to 21 Mar 2008,

Figure.6.16 The Floor Plan of the Test Apartment

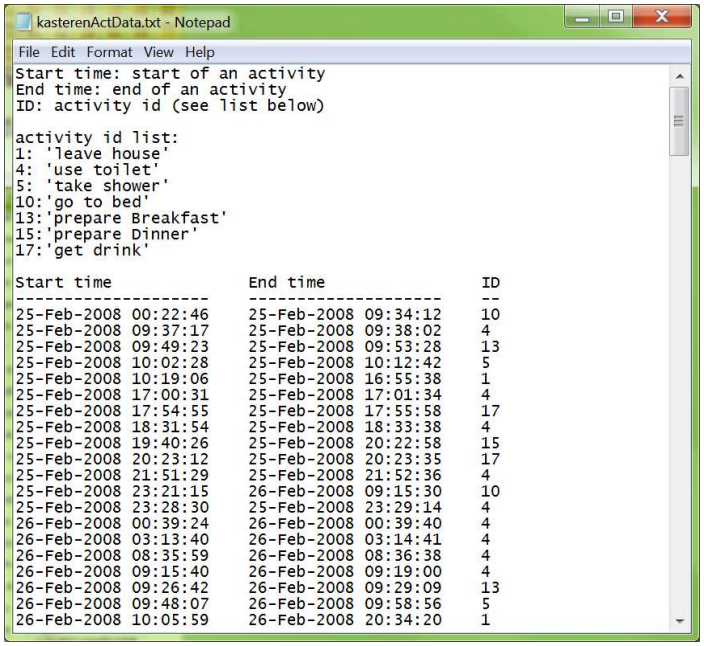
This annotated real world dataset contains 245 activities and corresponding states of binary inputs, as shown in Figure 6.17.

kasterenSenseData.txt - Notepad

File Edit Format View Help

```
Start time: Time sensor started firing
End time: Time sensor stopped firing
ID: sensor id (see list below)
val: value of sensor (always 1)

Sensor IDs are:
    [ 1]    'Microwave'
    [ 5]    'Hall-Toilet door'
    [ 6]    'Hall-Bathroom door'
    [ 7]    'Cups cupboard'
    [ 8]    'Fridge'
    [ 9]    'Plates cupboard'
    [12]    'Frontdoor'
    [13]    'Dishwasher'
    [14]    'ToiletFlush'
    [17]    'Freezer'
    [18]    'Pans Cupboard'
    [20]    'Washingmachine'
    [23]    'Groceries Cupboard'
    [24]    'Hall-Bedroom door'

Start time              End time                ID      Val
--------------------    --------------------    --      ---
25-Feb-2008 00:20:14    25-Feb-2008 00:22:57    24      1
25-Feb-2008 09:33:41    25-Feb-2008 09:33:42    24      1
25-Feb-2008 09:33:47    25-Feb-2008 17:21:12    24      1
25-Feb-2008 09:36:43    25-Feb-2008 09:37:04    5       1
25-Feb-2008 09:37:20    25-Feb-2008 09:37:23    6       1
25-Feb-2008 09:37:51    25-Feb-2008 09:37:52    14      1
25-Feb-2008 09:37:55    25-Feb-2008 09:37:56    14      1
25-Feb-2008 09:37:58    25-Feb-2008 09:38:01    6       1
25-Feb-2008 09:49:27    25-Feb-2008 09:49:28    9       1
25-Feb-2008 09:49:31    25-Feb-2008 09:49:38    9       1
25-Feb-2008 09:49:39    25-Feb-2008 09:49:44    8       1
25-Feb-2008 09:49:53    25-Feb-2008 09:49:56    8       1
```

(a) The State of 14 Binary Input Sensors



(b) User's Activities

Figure.6.17 Domestic Activity Dataset

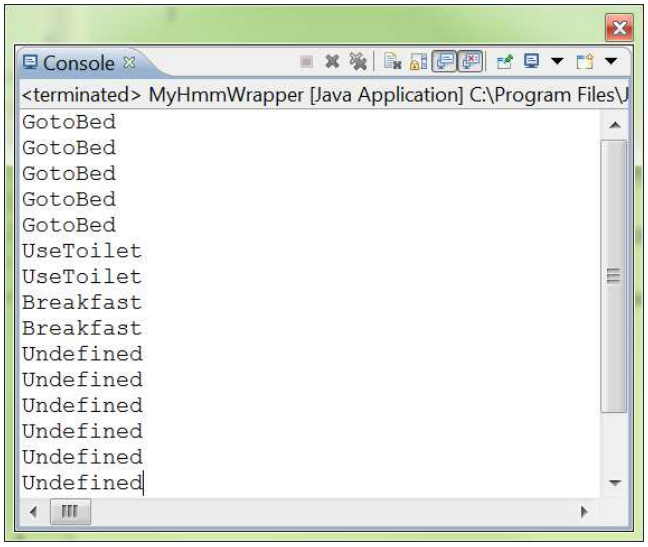
The task is split into two parts: estimating the process parameters from previous data (training data) and using these parameters to infer the real-world process by looking at the novel sensor reading. Based on Lim and Dey's work (Lim and Dey, 2010), we trained a HMM with a sequence length of 5 min, and 1 min per sequence step. The Baum-Welch algorithm (Baum et al., 1970) is used for training. In this subsection, we will not detail the mathematic train process. Detailed information on HMM can be consulted in (Rabiner, 1989).

The application takes 14 binary input sensors and infers which activity (out of seven) the user is performing. We focus only on verifying the HMM reasoner, thus sensor information is simulated in this example. We simulate a test sensor data sequence, which is represented by a 15×14 matrix. The row represents the situation of 14 sensors: "1" means that the sensor is sensed, while "0" means that the sensor is not sensed; the column represents the time slice. The data sequence is put in context-aware middleware successively. Then the learned HMM (parameters determined) reasoner infers the activity sequence by calculating its probability given an observation sequence (sensor data sequence). The Viterbi algorithm (Viterbi, 1967) is used to infer. The entire HMM process is implemented based on jhmm in Java. To make it easier to understand, the final result is

shown in Figure 6.18. As mentioned above, the HMM reasoner should work all the time. These results and historical results are stored in the context knowledge base, which can be used by various context-aware applications via context-aware middleware.

$$\begin{bmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&1\\
0&0&1&0&0&0&0&0&1&0&0&0&0&1\\
0&0&0&0&0&1&0&0&0&0&0&0&1&1\\
0&0&0&0&1&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{bmatrix}$$

(a) The simulated input data



(b) The printed results

Figure.6.18 The Results of HMM for Domestic Activity.

6.4 Analysis and Summary

In chapter two, the five representative five aspects of context-aware systems are provided, analyzed, and compared: architecture, context representation, context sensing, context storage, and context reasoning. We chose the main features for five aspects and compared our context-aware middleware with another five. The results are shown in Figure 6.19. The last row gives our middleware's performance.

System	Architecture			Context Representation			Context Sensing		Context Storage		Context Reasoning	
	Context Abstraction	Extensibility	Reusability	Context Model	Type of Context	Historical Support	Type of Sensing	Communication	Storage Database	Request Specification	Reasoning Support	Specific Reasoning
Context Tookit	+	+	+	Attribute Tuple	3	+	Sensor Nodes	TCP/IP	Attribute Tuple	SQL	-	-
CASS	+	+	+	Relational data	2	-	Sensor Nodes	Wi-Fi	Relational	SQL	+	-
CoBrA	+	+	+	Ontology-based	5	-	Context Acquisition Components	o	OWL	SPARQL	+	Rules
SOCAM	+	-	+	Ontology-based	5	+	Context Providers	RMI	OWL	SPARQL	+	Rules
CA-SOA	+	+	+	Ontology-based	5	o	Service Providers	Web Service	OWL	SPARQL	+	Rules
Ours	+	+	+	**Ontology-based**	**4**	+	**Context Providers**	**ESB**	**OWL**	**SPARQL**	+	**Ruls,DT,HMM**

Figure.6.19 Analysis and comparison with others

To be understood easily, we provide a quantitative analysis between the six context-aware systems shown in Figure 6.20. This analysis is based on a method that evaluates each feature as "weak, medium or strong" represented by the value: "1, 2, 3", then sums up all values of sub features as the final value. SOCAM's context representation is taken as an example to explain how to compute the feature's value. Context representation consists of three sub features: context model, type of context, and historical support. In the context model, it adopts the ontology-based model that is considered as "Strong" represented by value "3". It supports five main types of context, which is the maximum in six systems, considered as "Strong" represented by value "3". It provides historical context support viewed as "Strong" represented by value "3". The total value of context representation is "9", summed up by three sub features.

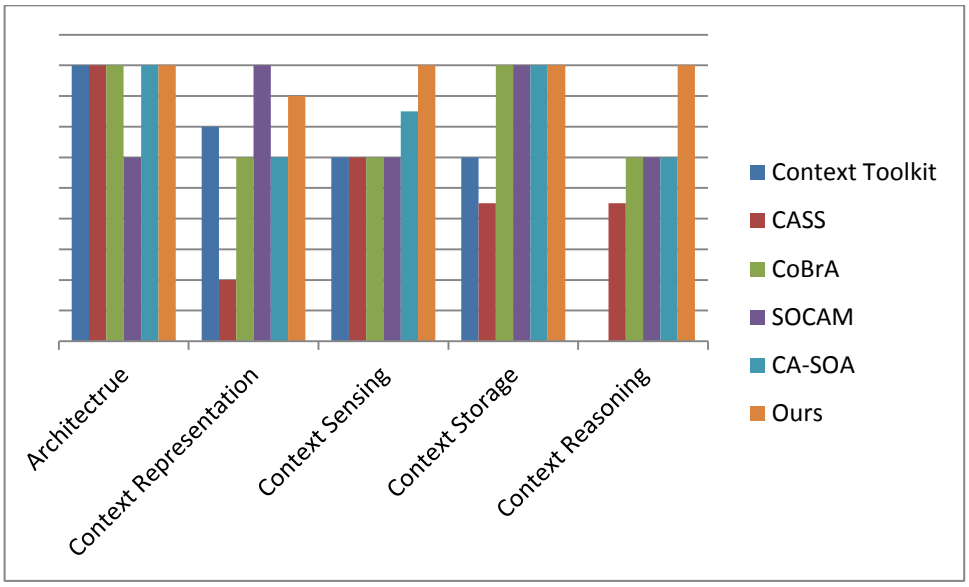


Figure.6.20 The quantitative analysis

In quantitative analysis, our middleware performs well with respect to architecture, supporting context abstraction, extensibility and reusability in a satisfactory manner. With respect to context representation, our middleware adopts the ontology-based model and supports historical context, but slightly fewer numbers of supported types of context than SOCAM and CA-SOA, so overall performance is a little weaker than SOCAM. With respect to context sensing, our middleware performs better than others since it supports more types of commutation by ESB. With respect to context storage, the main stream technology (OWL) is used in our middleware,

so its performance is no weaker than others. With respect to context reasoning, because we have designed an intelligent engine to support activity context recognition, it is much better than others.

To conclude, our middleware shows a good performance in five main aspects of context-aware systems, and especially in context reasoning. We will try to tackle this problem in a future study. As the application mainly involves user activities, our middleware will play an important role in context-aware application deployment.

However, these five aspects just represent the basic and fundamental requirements in context-aware systems. Many other aspects are involved and should be covered in terms of actual development, such as the issue of privacy protection that we have not taken into account for our middleware. We will try to tackle this problem in a future study. Through continuous improvement, our goal is to provide developers with a better platform for developing context-aware applications in ubiquitous computing environments.

6.5 Conclusions

In this chapter, the general process for building a context-aware application is provided first. Then, we proposed two scenarios (applications) to explain how to build a context-aware application, focusing on explaining how the inference engine is to be trained and work. Finally, we compare our context-aware middleware with another five representative context-aware systems described in chapter two. As our context-aware middleware benefits from good features from existing context-aware systems, we focus our research on improving intelligence of context-aware systems via supporting activity context recognition. Its performance is thus superior to others in context reasoning.

7 Conclusion and Future work

7.1 Introduction

In this last chapter we summarize the contributions of the thesis and then briefly discuss future directions.

7.2 Contributions

The goal of this research is to propose a system architecture that facilitates the development of context-aware applications. The main research contributions are presented in three categories: a spatial-temporal context representation model, a context-aware middleware, and an intelligence context inference engine.

- **The spatial-temporal context representation model**

  With respect to context representation, a spatial-temporal context representation model is proposed for our context-aware middleware via ontology. This model adopts a two-layer hierarchy structure for modeling context in both spatial dimension and temporal dimension. The higher layer deals with the generic common context, consisting of five interrelated basic classes: user, location, time, activity, and device, while the lower layer is used to complete and upgrade the higher layer by more precise information related to a particular application or application area. Unlike existing models, besides taking location factors into account, our context model also ranges contextual information during periods. These historical contexts may be used to predict and infer the context. In addition, the ontology-based modeling approach facilitates development of the context-aware system, such as knowledge sharing, knowledge reuse, and logic inference.

- **A context-aware middleware in Ambient Intelligence**
  To implement this more in-depth approach to context-aware services in AmI, we designed a context-aware middleware based on a spatial-temporal context representation model, organized in two layers. The low layer is an Enterprise Service Bus, which provides a solution for integrating sensors and actuators with a standardized data representation and unified standard interface to achieve the core functions of service interaction: service registry, service discovery, and service consumption. The high layer is a versatile context interpreter in charge of context inference, expressive query, and persistent storage. It consists of four parts: the context aggregator, the context knowledge, the context query engine, and the intelligent inference engine that, in its role as the brain for context-aware middleware, is another focus of research.

- **The intelligence context inference engine**

  The third contribution in this dissertation focuses on designing a context inference engine, supporting activity context recognition to improve context-aware middleware intelligence. We review all the methods for activity context recognition published in three premier conferences in the past decade and conclude that activity context recognition is divided into three facets: basic activity inference, dynamic activity analysis, and future activity recommendation. We propose an intelligent inference engine based on our context-aware middleware. This engine integrates the three most popular methods for activity context recognition used in context-aware applications: Rules, the Decision Tree, and the Hidden Markov Model, via a strategy pattern. Along with the inference engine, an associated invoking mechanism is designed to cope with different tasks. This intelligent inference engine not only provides a solution to meet current requirements in activity context recognition, but also has the extension ability to deal with future requirements.

7.3 Future works

My future work plans to investigate the cross-domain combination of ubiquitous computing and big data research based on our system via machine learning. “Big Data” is currently one of the most popular terms in Computer Science, considered as the next frontier for innovation. To some extent, Big Data are growing thanks to ubiquitous computing. Advances in digital

sensors, communications, computation, and storage have created huge collections of data, capturing information from business, science and society (Bryant et al., 2008) The remainder of this section describes two potential research directions.

Ubiquitous Analytics

The rise of big data increasingly requires that we are able to access data resources anytime and anywhere. These massive data volumes are gathered from multiple heterogonous resources in various forms, including diversity sensors, retailers and even billions of individuals' social networks. Before benefiting from the full profits of big data, many issues must be dealt with, such as data access, data fusion and integration. Our system can provide some efficient solutions for this.

Activity prediction based on Big Data

As the central theme in ubiquitous computing, Context Awareness's Holy Grail is to divine or understand human intent. Thanks to massive amounts of data about users, researchers are able to predict users' behavior trends. Moreover, aggregating and analyzing individual data can provide personalized service, targeted recommendation and prediction. One of the most interesting stories to come out of the presidential election was the use of big data analytics to help campaign workers bring out the vote. As mentioned in chapter 5, we have made attempts, employing individuals' activity information gathered from their social networks, to construct the intelligent inference engine.

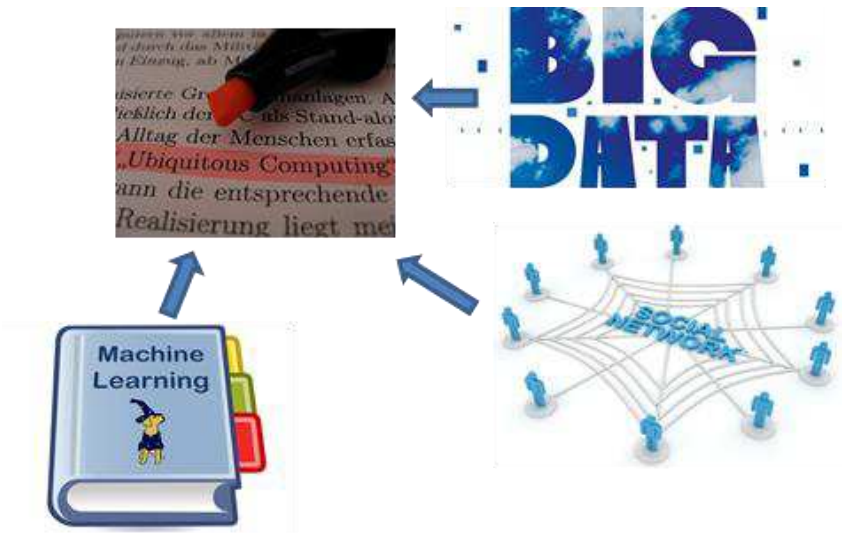


Figure.7.1. The new ubiquitous computing technologies

7.4 Conclusions

This dissertation has investigated and implemented a context-aware middleware for developing context-aware applications in ambient intelligence. Research work focuses on three aspects: context model, middleware architecture, and context inference. The spatial-temporal context representation model paves the way for the whole system. The context-aware middleware provides a feasible solution for rapid development of new applications in ambient intelligence. The intelligent inference engine improves context-aware middleware intelligence by supporting activity context recognition.

Bibliographie

Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M., Steggles, P., 1999. Towards a Better Understanding of Context and Context-Awareness, in: Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing, HUC'99. Springer-Verlag, London, UK, UK, pp. 304–307.

Abox, 2009. . Wikipedia Free Encycl. Available from: http://en.wikipedia.org/wiki/Abox

Ambient intelligence, 2013. . Wikipedia Free Encycl. Available from: http://en.wikipedia.org/wiki/Ambient_intelligence

Apache Jena, 2010. Available from: http://jena.apache.org/

Athanasopoulos, D., Zarras, A.V., Issarny, V., Pitoura, E., Vassiliadis, P., 2008. CoWSAMI: Interface-aware context gathering in ambient intelligence environments. Pervasive Mob Comput 4, 360–389.

Baldauf, M., Dustdar, S., Rosenberg, F., 2007a. A survey on context-aware systems. Int J Ad Hoc Ubiquitous Comput 2, 263–277.

Baldauf, M., Dustdar, S., Rosenberg, F., 2007b. A survey on context-aware systems. Int J Ad Hoc Ubiquitous Comput 2, 263–277.

Baum, L., Petrie, T., Soules, G., Weiss, N., 1970. A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. Ann. Math. Stat. 41, 164–171.

Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., Riboni, D., 2010. A survey of context modelling and reasoning techniques. Pervasive Mob. Comput. 6, 161–180.

Bhat, M., 2008. SOA? ESB? What is all this? [WWW Document]. Available from: http://software.intel.com/en-us/articles/soa-esb-what-is-all-this

Borst, W., 1997. Construction of Engineering Ontologies for Knowledge Sharing and Reuse. Ph.D thesis, University of Twente, Netherlands

Breiman, L., 1984. Classification and regression trees. Wadsworth International Group.

Brown, P.J., 1998. Triggering information by context. Pers. Technol. 2, 18–27.

Brown, P.J., Bovey, J.D., Chen, X., 1997. Context-aware applications: from the laboratory to the marketplace. IEEE Pers. Commun. 4, 58–64.

Bryant, R., Katz, R.H., Lazowska, E.D., 2008. Big-Data Computing: Creating Revolutionary Breakthroughs in Commerce, Science and Society. In Computing Research Initiatives for the 21st Century. Available from http: //www.cra.org/ccc/initiatives

Chen, H., Finin, T., Joshi, A., 2003. An Intelligent Broker for Context-Aware Systems. Adjun. Proc. Ubicomp, 12–15.

Chen, Harry, Finin, T., Joshi, A., 2004. A Context Broker for Building Smart Meeting Rooms. Proceedings of the AAAI Symposium on Knowledge Representation and Ontology for Autonomous Systems Symposium, 2004 AAAI Spring Symposium, 53–60

Chen, H., Finin, T., Joshi, A., Kagal, L., Perich, F., Chakraborty, D., 2004. Intelligent agents meet the semantic Web in smart spaces. IEEE Internet Comput. 8, 69–79.

Chen, I.Y.L., Yang, S.J.-H., Zhang, J., 2006. Ubiquitous Provision of Context Aware Web Services, in: IEEE International Conference on Services Computing, 2006. SCC'06. Presented at the IEEE International Conference on Services Computing, 2006. SCC'06, 60–68.

DAML+OIL, 2001. Available from: http://www.w3.org/TR/daml+oil-reference

De Almeida, D.R., de Souza Baptista, C., da Silva, E.R., Campelo, C.E.C., de Figueiredo, H.F., Lacerda, Y.A., 2006. A context-aware system based on service-oriented architecture, in: 20th International Conference on Advanced Information Networking and Applications, 205-210.

Dey, A.K., 2000. Providing architectural support for building context-aware applications. Ph.D thesis, Georgia Institute of Technology, Atlanta, GA, USA.

Dey, A.K., Abowd, G.D., 1999a. Towards a better understanding of context and context-awareness, in: In HUC '99: Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing. Springer-Verlag, 304–307.

Dey, A.K., Abowd, G.D., 1999b. The Context Toolkit: Aiding the Development of Context-Aware Applications. Proc. Workshop Software Eng. for Wearable and Pervasive Computing, ACM Press, New York, 434–441

Dey, A.K., Abowd, G.D., Salber, D., 2001. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Hum-Comput Interact 16, 97–166.

Elena, D.P., Not, E., Stock, O., Zancanaro, M., 2000. Modeling Context Is Like Taking Pictures. In Proc. of the What, Who, Where, When, Why and How of Context-Awareness Workshop

Esler, M., Hightower, J., Anderson, T., Borriello, G., 1999. Next century challenges: Data-centric networking for invisible computing – The Portolano Project at the University of Washington, 256–262.

Fahy, P., Clarke, S., 2004. CASS – a middleware for mobile context-aware applications, in: Workshop on Context Awareness, MobiSys, USA.

Fickas, S., Kortuem, G., Segall, Z., 1997. Software organization for dynamic and adaptable wearable systems, in: First International Symposium on Wearable Computers, 56–63.

Floréen, P., Przybilski, M., Nurmi, P., Koolwaaij, J., Tarlano, A., Wagner, M., Luther, M., Bataille, F., Boussard, M., Mrohs, B., Lau, S., 2005. Towards a Context Management Framework for MobiLife, in: In IST Mobile & Wireless Communications Summit. Germany.

Franklin, D., Flachsbart, J., 1998. All gadget and no representation makes Jack a dull environment. In AAAI Spring Symposium on Intelligent Environments, 155–160

Freeman, Eric, Freeman, Elizabeth, Sierra, K., Bates, B., 2004. Head first design patterns. O'Reilly, Sebastopol.

Fu, Z., Yue, J., Li, Z., 2009. Ontology and Its Application in Supply Chain Information Management, in: Huo, Y., Ji, F. (Eds.), Supply Chain the Way to Flat Organisation. InTech.

Geib, C.W., Maraist, J., Goldman, R.P., 2008. A New Probabilistic Plan Recognition Algorithm Based on String Rewriting, in: ICAPS. 91–98.

Gruber, T.R., 1993. A translation approach to portable ontology specifications. Knowl. Acquis. 5, 199–220.

Gu, T., Pung, H.K., Yao, J.K., 2005a. Towards a Flexible Service Discovery. Elsevier J. Netw. Comput. Appl. JNCA 28, 233–248.

Gu, T., Pung, H.K., Zhang, D.Q., 2004. A middleware for building context-aware mobile services, in: Vehicular Technology Conference. 2656–2660.

Gu, T., Pung, H.K., Zhang, D.Q., 2005b. A service-oriented middleware for building context-aware services. J Netw Comput Appl 28, 1–18.

Gwizdka, J., 2000. What's in the context? Modeling Context-Aware Systems, 1–4

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H., 2009. The WEKA data mining software: an update. SIGKDD Explor Newsl 11, 10–18.

Han, B., Jia, W., Shen, J., Yuen, M.-C., 2005. Context-Awareness in Mobile Web Services, in: Cao, J., Yang, L.T., Guo, M., Lau, F. (Eds.), Parallel and Distributed Processing and Applications, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 519–528.

Henricksen, K., Indulska, J., Rakotonirainy, A., 2003. Generating Context Management Infrastructure from High-Level Context Models, in: In 4th International Conference on Mobile Data Management (MDM) - Industrial Track. 1–6.

Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K., 2000. System architecture directions for networked sensors, in: Architectural Support for Programming Languages and Operating Systems. 93–104.

Hofer, T., Schwinger, W., Pichler, M., Leonhartsberger, G., Altmann, J., Retschitzegger, W., 2003. Context-awareness on mobile devices - the hydrogen approach, in: Proceedings of the 36th Annual Hawaii International Conference on System Sciences, 292–302

Horn, A., 1951. On Sentences Which are True of Direct Unions of Algebras. J. Symb. Log. 16, 14–21.

Hu, D.H., Zheng, V.W., Yang, Q., 2011. Cross-domain activity recognition via transfer learning. Pervasive Mob Comput 7, 344–358.

Hull, R., Neaves, P., Bedford-Roberts, J., 1997. Towards situated computing, in: , First International Symposium on Wearable Computers, 1997. Digest of Papers. Presented at the , First International Symposium on Wearable Computers,146–153.

Johanson, B., Fox, A., Winograd, T., 2002. The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. IEEE Pervasive Comput. 1, 67–74.

Kass, G.V., 1980. An Exploratory Technique for Investigating Large Quantities of Categorical Data. Appl. Stat. 29, 119–127.

Keidl, M., Kemper, A., 2004. Towards context-aware adaptable web services, in: Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers &amp, 55–65.

Kindberg, T., Barton, J., Morgan, J., Becker, G., Caswell, D., Debaty, P., Gopal, G., Frid, M., Krishnan, V., Morris, H., Schettino, J., Serra, B., Spasojevic, M., 2000. People, places, things: Web presence for the real world, in: Mobile Computing Systems and Applications, Third IEEE Workshop On. Presented at the Mobile Computing Systems and Applications, 19–28.

Kjær, K.E., 2007. A survey of context-aware middleware, in: Proceedings of the 25th Conference on IASTED International Multi-Conference: Software Engineering, SE'07. ACTA Press, Anaheim, CA, USA, 148–155.

Korpipaa, P., Mantyjarvi, J., Kela, J., Keranen, H., Malm, E.-J., 2003. Managing Context Information in Mobile Devices. IEEE Pervasive Comput. 2, 42–51.

Krumm, J. (Ed.), 2009. Ubiquitous Computing Fundamentals. Chapman and Hall/CRC.

Lim, B.Y., Dey, A.K., 2010. Toolkit to support intelligibility in context-aware applications, in: Proceedings of the 12th ACM International Conference on Ubiquitous Computing, Ubicomp'10. ACM, New York, NY, USA, 13–22.

Lin, L., 2006. Location-based activity recognition, Ph.D thesis, University of Washington.

McCarthy, J., 1993. Notes on formalizing context, in: Proceedings of the 13th International Joint Conference on Artifical Intelligence - Volume 1, IJCAI'93. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 555–560.

Melchiorre, C., Reiff-Marganiec, S., Yu, H.Q., Shall, D., Dustdar, S., 2008. InContext - Interaction and Context-based Technologies for Collaborative Teams. IOS Press, Amsterdam.

Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., Swartout, W.R., 1991. Enabling technology for knowledge sharing. AI Mag 12, 36–56.

OWL Web Ontology Language, 2004. Available from: http://www.w3.org/TR/owl-features/

Pallapa, G.V., 2010. A privacy enhanced situation-aware middleware framework for ubiquitous computing environments. Ph.D thesis, University of Texas at Arlington, Arlington, TX, USA.

Pascoe, J., Ryan, N.S., Morse, D.R., 1998. Human Computer Giraffe Interaction: HCI in the Field, in: Workshop on Human Computer Interaction with Mobile Devices, University of Glasgow, United Kingdom, 21- 23 May 1998, GIST Technical Report G98-1

Pazzani, M., Muramatsu, J., Billsus, D., 1996. Syskill & Webert: Identifying interesting web sites, in: Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 1, AAAI'96. AAAI Press, 54–61.

Pollack, M.E., Brown, L., Colbry, D., McCarthy, C.E., Orosz, C., Peintner, B., Ramakrishnan, S., Tsamardinos, I., 2003. Autominder: An Intelligent Cognitive Orthotic System for People with Memory Impairment. Robotics and Autonomous Systems, Volume 44, Issues 3–4, 273–282

Protégé, 2004. Available from: http://protege.stanford.edu/

Quinlan, J.R., 1986. Induction of decision trees. Mach. Learn. 1, 81–106.

Quinlan, J.R., 1993. C4.5: programs for machine learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Rabiner, L., 1989. A tutorial on hidden Markov models and selected applications in speech recognition. in Proc. IEEE , Volume 77, Issue 2, 257–286.

Ranganathan, A., Campbell, R.H., 2003. A Middleware for Context-Aware Agents in Ubiquitous Computing Environments, in: Endler, M., Schmidt, D. (Eds.), Middleware 2003, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 143–161.

Resource Description Framework, 2004. Available from: http://www.w3.org/RDF/

Román, M., Hess, C., Cerqueira, R., Campbell, R.H., Nahrstedt, K., 2002a. Gaia: A Middleware Infrastructure to Enable Active Spaces. IEEE Pervasive Comput. 1, 74–83.

Román, M., Hess, C., Cerqueira, R., Campbell, R.H., Nahrstedt, K., 2002b. Gaia: A Middleware Infrastructure to Enable Active Spaces. IEEE Pervasive Comput. 1, 74–83.

Romero, D., Parra, C., Seinturier, L., Duchien, L., Casallas, R., 2008. An SCA-Based Middleware Platform for Mobile Devices, in: Enterprise Distributed Object Computing Conference Workshops, 2008 12th. Presented at the Enterprise Distributed Object Computing Conference Workshops, 2008 12th, pp. 393–396.

Rudolph, L., 2001. Project Oxygen: Pervasive, Human-Centric Computing - An Initial Experience, in: Proceedings of the 13th International Conference on Advanced Information Systems Engineering, CAiSE '01. Springer-Verlag, London, UK, UK, pp. 1–12.

Ryan, N., 1997. MCFE metadata elements, version 0.2. Working document. Available from: http://www.cs.kent.ac.uk/projects/mobicomp/Fieldwork/Notes/mcfemeta.html

Ryan, N., Pascoe, J., Morse, D., 1997. Enhanced Reality Fieldwork: the Context-aware Archaeological Assistant, in: British Archaeology Report, Oxford.

Salber, D., Dey, A.K., Abowd, G.D., 1999. The context toolkit: aiding the development of context-enabled applications, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI'99. ACM, New York, NY, USA, 434–441.

Salerno, S., 2008. The Learning Grid Handbook: Concepts, Technologies and Applications. IOS Press.

Satyanarayanan, M., 2001. Pervasive computing: vision and challenges. IEEE Pers. Commun. 8, 10–17.

Schilit, B., Adams, N., Want, R., 1994. Context-Aware Computing Applications, in: In Proceedings of the Workshop on Mobile Computing Systems and Applications. IEEE Computer Society, pp. 85–90.

Schilit, B.N., Theimer, M.M., 1994. Disseminating active map information to mobile hosts. IEEE Netw. 8, 22–32.

Schmidt, A., Laerhoven, K.V., 2001. How to build smart appliances? IEEE Pers. Commun. 8, 66–71.

Sheng, Q.Z., Benatallah, B., 2005. ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services Development, in: Proceedings of the International Conference on Mobile Business, ICMB '05. IEEE Computer Society, Washington, DC, USA, 206–212.

Singh, A., Conway, M., 2006. Survey of Context aware Frameworks - Analysis and Criticism (Technical report). University of North Carolina.

Sørensen, C.-F., Wu, M., Sivaharan, T., Blair, G.S., Okanda, P., Friday, A., Duran-Limon, H., 2004. A context-aware middleware for applications in mobile Ad Hoc environments, in: Proceedings of the 2nd Workshop on Middleware for Pervasive and Ad-hoc Computing, MPAC '04. ACM, New York, NY, USA, 107–110.

Sousa, J.P., Sousa, J.P., Garlan, D., Garlan, D., 2002. Aura: an architectural framework for user mobility in ubiquitous computing environments, in: In Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture. Kluwer Academic Publishers, 29–43.

SPARQL Query Language for RDF, 2008. Available from: http://www.w3.org/TR/rdf-sparql-query/

Strang, T., Linnhoff-Popien, C., 2004. A Context Modeling Survey, in: In: Workshop on Advanced Context Modelling, Reasoning and Management, The Sixth International Conference on Ubiquitous Computing, Nottingham/England.

Studer, R., Benjamins, V.R., Fensel, D., 1998. Knowledge engineering: Principles and methods. Data Knowl. Eng. 25, 161–197.

Tbox, 2009. Wikipedia Free Encycl. Available from: http://en.wikipedia.org/wiki/Tbox

Truong, H., Juszczyk, L., Manzoor, A., Dustdar, S., 2007. Escape - an adaptive framework for managing and providing context information in emergency situations, in: Second European Conference, EuroSSC 2007, Kendal, England, 207–222.

Truong, H.-L., Dustdar, S., 2009. A survey on context-aware web service systems. Int. J. Web Inf. Syst. 5, 5–31.

Vail, D.L., Veloso, M.M., Lafferty, J.D., 2007. Conditional random fields for activity recognition, in: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, 235:1–235:8.

Van Kasteren, T., Noulas, A., Englebienne, G., Kröse, B., 2008. Accurate activity recognition in a home setting, in: Proceedings of the 10th International Conference on Ubiquitous Computing, UbiComp'08. ACM, New York, NY, USA, 1–9.

Viterbi, A.J., 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Trans. Inf. Theory 13, 260–269.

Wang, X.H., Zhang, D.Q., Gu, T., Pung, H.K., 2004. Ontology Based Context Modeling and Reasoning using OWL, in: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, 18–23.

Want, R., Hopper, A., Falcão, V., Gibbons, J., 1992a. The active badge location system. ACM Trans Inf Syst 10, 91–102.

Want, R., Schilit, B.N., Adams, N.I., Gold, R., Petersen, K., Goldberg, D., Ellis, J.R., Weiser, M., 1995. An overview of the PARCTAB ubiquitous computing experiment. IEEE Pers. Commun. 2, 28–43.

XML, 2013. . Wikipedia Free Encycl. Available from: http://en.wikipedia.org/wiki/XML

Xu, T., David, B., Chalon, R., Zhou, Y., 2011. A context-aware middleware for ambient intelligence, in: Proceedings of the Workshop on Posters and Demos Track, 10:1–10:2.

Publications

Xu, T., Zhou, Y., David, B., Chalon, R., Supporting Activity Context Recognition in Context-aware Middleware, Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI'13), Bellevue, Washington, USA. (2013)

Xu, T., Jin, H., David, B., Chalon, R., Zhou, Y., A context-ware middleware for interaction devices deployment in AmI, the 15th International Conference on Human-Computer Interaction (HCI'13), Las Vegas, Nevada, USA (2013)

Xu, T., David, B., Chalon, R., Zhou, Y., A Context-aware Middleware for Ambient Intelligence. ACM/IFIP/USENIX 12th International Middleware Conference (Middleware 2011), Poster session, Lisbon, Portugal, 2011.

[Journal] *Zhou, Y., Xu, T., David, B., Chalon, R., Innovative Wearable Interfaces: An Exploratory Analysis of Paper-based Interfaces with Camera-glasses Device Unit, Journal of Personal and Ubiquitous Computing, to appear, 2013. (SCI IF: 0.938)*

David, B., Xu, T., Jin, H., Zhou, Y., Chalon, R., Zhang, B., Yin, C., Wang, C., User-oriented System for Smart City approaches, HMS 2013, Las Vegas, Nevada, USA (2013)

Jin, H., Xu, T., David, B., Chalon, R., Zhou, Y., Proxemic Interaction Applied to Public Screen in Lab, HCI'13, Las Vegas, Nevada, USA (2013 Accepted)

David, B., Yin, C., Zhou, Y., Xu, T., Zhang, B., Jin, H., Chalon, R., SMART-CITY: problematic, techniques and case studies. 8th International Conference on Computing Technology and Information Management (ICCM 12), Korea. (2012)

David, B., Zhou, Y., Xu, T., Chalon, R., Mobile User Interfaces and their Utilization in a Smart City, ICOMP'11, Las Vegas, Nevada, USA. pp. 383-388. (2011)

Abstract

Almost 20 years ago, Marc Weiser envisioned the prospect of computer in 21st century, and proposed the pioneering notion of ubiquitous computing. One of Weiser's primary ideas has recently evolved to a more general paradigm known as context awareness, becoming a central research theme in many other ubiquitous computing programs. From Active Badge considered as the first context-aware application, there are numerous attempts to build effective context-aware systems. However, how to acquire context, how to process context and how to create context-aware applications is still faced with enormous challenges in the both of research and practice.

This dissertation investigates deeply some chosen key issues in context awareness and develops a context-aware middleware. The main research contributions are presented in three categories: a spatial-temporal context represent model, a context-aware middleware and an intelligence context inference engine.

The spatial-temporal context representation model is proposed to organize context and relations for context-aware system. Ontology-based method is adopted to construct our model, supporting both knowledge sharing and reuse as well as logic inference. This model adopts two-layer hierarchy structure for different situation. The higher layer comes up with the generic common context, while the lower layer focuses on various specific situations. Differing from existing models, besides taking locational factors into account, it supports different historical context service depending on different context resource. These context histories may be used to predict and infer the context.

A context-aware middleware is designed as a platform associated with context retrieval and context processing. It is organized in two layers: the low layer provides a solution to integrate sensors and actuators with a standardized data representation; the high layer: versatile context interpreter focuses on context processing, which is made up of four parts: Context Aggregator, Inference Engine, Context Knowledge Base, and Query Engine in charge of context inferences, expressive query, and persistent storage. This middleware provides an environment for rapid prototyping of context aware services in ambient intelligent.

The intelligent inference engine is the central and intellectual component of context-aware middleware. We review all the methods on activity context recognition published in three premier conferences in past decade and conclude that activity context recognition is divided into three facets: basic activity inference, dynamic activity analysis and future activity recommendation. Then we propose an intelligent inference engine based on our context-aware middleware. Beside satisfying requirements of checking the context consistency, our inference engine integrates the three most popular methods on activity context recognition: Rules, Decision Tree, and Hide Markov Model. It provides a solution for all facets of activity context recognition based on our context-ware middleware. The individuals' information collecting from their social networks under permission are leveraged to train intelligent inference engine.

We finally use two scenarios (applications) to explain the generic process to develop application via our middleware, and compare and analyze the main aspects of our middleware with other five representative context-aware applications. Our middleware profits good features from existing context-aware systems and improve intelligence via supporting activity context recognition. It provides an efficient platform for a rapid developing of new context-aware applications in ambient intelligence.

Key words: Ubiquitous Computing, Context Awareness, Ambient Intelligence, Middleware

Résumé

Il y a près de 20 ans, Marc Weiser a imaginé l'ordinateur du 21ème siècle et a proposé le concept de l'informatique ubiquitaire. Une des idées principales de Weiser a récemment évolué vers un paradigme plus général connu comme la sensibilité au contexte qui est devenu un thème très important en informatique ubiquitaire. Depuis *Active Badge* considéré comme la première application sensible au contexte, de nombreuses tentatives pour construire des systèmes sensibles au contexte efficaces ont vu le jour. Cependant la problématique comment acquérir contexte, comment le traiter et comment créer des applications sensibles au contexte est encore aujourd'hui un défi suscitant de nombreuses recherches.

Cette thèse étudie en profondeur certaines questions clés liées à la sensibilité au contexte et au développement d'intergiciels sensibles au contexte. Les principales contributions de notre recherche concernent la prise en compte du contexte spatio-temporel et sa modélisation, la conception et l'implémentation d'un intergiciel sensible au contexte et d'un moteur intelligent d'inférence de contexte.

Le modèle de représentation du contexte spatio-temporel proposé vise à organiser le contexte et ses relations pour un système sensible au contexte. La méthode basée sur les ontologies est adoptée pour construire notre modèle, supportant à la fois le partage des connaissances et leur réutilisation ainsi que la déduction logique. Ce modèle adopte une structure hiérarchique à deux couches pour modéliser les situations à prendre en compte. La couche supérieure s'occupe du contexte commun générique, tandis que la couche inférieure se concentre sur les caractéristiques plus spécifiques. A la différence des modèles existants, en plus de prendre en compte l'aspect localisation, notre modèle prend également en charge la gestion d'historique des de contextes en s'appuyant sur différentes ressources. Ces historiques de contexte peuvent être utilisées pour prévoir et inférer le contexte.

Un middleware sensible au contexte a été conçu comme une plate-forme permettant la récupération et le traitement du contexte. Elle est organisée en deux couches : La couche basse apporte une solution à l'intégration des capteurs et actionneurs avec une représentation de données normalisée ; la couche haute propose un interpréteur de contexte polyvalent qui s'appuie sur quatre éléments : un agrégateur de contexte, un moteur d'inférence, une base de connaissance de contextes et un moteur de recherche en charge de la déduction de contexte, de l'interrogation et du stockage persistant.

Ce middleware fournit un environnement pour le prototypage rapide de services sensibles au contexte pour l'intelligente ambiante.

Le moteur intelligent d'inférence est le composant central de notre middleware. Pour le concevoir nous avons d'abord étudié toutes les méthodes publiées pendant les dix dernières années dans les trois conférences de premier plan du domaine. Nous en avons retiré que la reconnaissance du contexte d'activité peut être obtenue par trois catégories de traitements : par l'activité de l'inférence de base, par l'analyse dynamique de l'activité et par la recommandation d'activités futures. Nous proposons alors un moteur d'inférence intelligent basé sur notre middleware sensible au contexte. Outre les exigences liées à la vérification de la cohérence du contexte, notre moteur d'inférence intègre les trois méthodes les plus populaires concernant la reconnaissance de contexte : des règles, des arbres de décision, et les Modèles de Markov Cachés. Ceci constitue une solution intéressante couvrant toutes les facettes de l'activité de reconnaissance de contexte dans notre middleware sensible au contexte. Les informations collectées à partir des réseaux sociaux sont utilisées pour éduquer le moteur d'inférence intelligent.

Nous utilisons deux scénarios (applications) pour concrétiser le processus générique et pour développer des applications à l'aide de notre middleware, notamment pour comparer et pour analyser les principaux aspects de notre middleware avec cinq autres applications représentatives sensibles au contexte.

Notre middleware intègre des bonnes caractéristiques issues des systèmes existants et améliore l'intelligence en soutenant l'activité de reconnaissance de contexte. Il fournit une plate-forme efficace pour un développement rapide de nouvelles applications sensibles au contexte dans le domaine de l'intelligence ambiante.

Mots clés : Informatique Ubiquitaire, Sensibilité au Contexte, Intelligence Ambiante, Middleware