Design of Physical and Logical Context Aware Middleware

Junzhong Gu[1] and Gong-Chao Chen[2]

[1]Institute of Computer Applications, East China Normal University, 200062 Shanghai, China
[2]School of Information Security Engineering, Shanghai Jiao Tong University, Shanghai 200240, China
jzgu@ica.stc.sh.cn, 18918580100@189.cn

Abstract

The human-computer interaction depended computing mode is evolved to an intelligent adaptive context aware computing. The corresponding middleware should be adaptive, reflective, dynamic reconfigurable and physical/virtual context aware. In this paper the design of a context-aware middleware is discussed. Some key issues involved, e.g. virtual and meta sensors, reflective context model, multi-agent mechanism, an extension of Web Service-Open Web Service, etc. are analyzed and studied. An experimental system and its design are introduced.

Keywords: *physical context awareness, logical context awareness, middleware, mobile agent, virtual sensor, meta sensor, reflective system, OpenWebService*

1. Introduction

Boundary of Internet is widely extended. Connected devices are extended from workstations/PCs to handhelds, as well as various sensors. IOT[1] is one of such cases. The fixed network based computing is evolved to a mobile environment. The human-computer interaction depended computing mode is evolved to intelligent adaptive context aware computing. [1, 2, 3] Therefore intelligent agents, especially mobile agents, are in good grace. With characters such as distributed and mobile environment, heterogeneity, context awareness, physical/logical sensor orientation, and so on, developing responsible middleware system is urgently demanded. Different from traditional middleware frameworks such as message oriented middleware, object-oriented middleware, etc., context aware middleware should be adaptive, intelligent, reflective and dynamic reconfigurable.

Some common factors affect the design of the middleware infrastructure in context aware computing, such as: (Mobile) Devices, Reusable and adaptive, Communication and Privacy.

Devices: Things are introduces to Internet. They can be categorized to two classes, computable (devices) and computing enable things. Computing enable things refer those without computing power, such as things (e.g. books, parcels, etc.) are tagged with RFID[2]. They connect to Internet via special gateways (e.g. RFID readers). Such a gateway plays a role of computable device. The computable devices are with computing power, such as PCs,

1 IOT-Internet of Things
2 Radio-frequency identification (RFID) is the use of a wireless non-contact radio system to transfer data from a tag attached to an object, for the purposes of automatic identification and tracking.

PDAs, smart phones, as well as online gateways (e.g. RFID readers) serving computing enable things.

Devices, mainly mobile, vary from one to another in term of resource availability. For example, laptops are different from PDAs and smart phones in CPU computing power, and amount of RAM and disk space. Hence, the corresponding middleware should be designed to flexibly sense context via various sensors and automatically optimize resource utilization. Mobile clients may interact with different types of networks, services, and security policies as they move from one area to another.

Mobile devices (including online gateways serving enable things) often hinder the functionality of their respective systems, making difficult to build more advanced and complex system behaviors, for the limited computational resources, such as battery power, information storage, computing power, and communication:

Battery Power Constraint-Context-aware agents often rely on sensors to acquire contexts. Most of the mobile devices do not support sophisticated sensing hardware due to their battery power constraint.

Information Storage Constraint-Context-aware agents often need to store acquired contextual knowledge to reduce the cost of performing repetitive context acquisition and to exploit the historical information of contexts. Limited by the disk space on mobile devices, agents usually do not have the privilege to continuously store all knowledge. Agents will be faced with the challenge to be selective about what kind of knowledge should be stored and what kind of knowledge should be deleted. Contextual knowledge acquired from physical environment is potentially inaccurate, agents will be challenged to maintain knowledge consistency and resolve information ambiguity.

Computing Power Constraint- Because mobile devices are often lack of enough CPU power and memory, context-aware agents on mobile devices can only process primitive contexts. Even though computation is done at background, it raises the issues in the communication constraint of mobile devices.

Due to these limitations, conventional middleware technologies designed for fixed distributed systems are not suitable.

Reusable and adaptive -Context sensing and reasoning are two important steps in context aware applications [3].

Context sensing is to acquire information via hardware/logical (software) sensors. Facing on a dynamic environment, a great amount of hardware/software sensors are required for agents to acquire a wide divergence of information. Without reusable and adaptive mechanisms, directly attach sensors with context-aware agents also makes difficult for building more advanced and complex systems. Therefore, context aware middleware should be reusable and adaptive.

Context reasoning mechanisms varies from simple intelligent query to complex rule-based programming. With the increasing demand to process more and more advanced contexts, context reasoning mechanisms will become more complex and specialized. Without any reusable and adaptable mechanisms for context reasoning, building context-aware systems from scratch will be difficult and costly.

Communication Issue- Previous middleware is always fixed connection oriented, where RPC[3], object requests, RMI[4] or distributed transactions assume a high-bandwidth connection of the components, as well as their constant availability. In mobile systems unreachability and low bandwidth are the norm rather than a failure.

[3] Remote Procedure Call
[4] Remote Method Invocation

Mobile applications should replicate information in order to access them off-line. Replication needs synchronization when a connection is re-established. The commonly used principle of transparency prevents the middleware to exploit knowledge that only the application has, such as which portion of data to replicate and which reconciliation policy to apply.

In a dynamic environment, the communication between agents on mobile devices and the nearby context sources often cannot be pre-defined. Agents may be lack of sufficient knowledge to communicate with context sources (e.g. knowing which sensor can provide what information and how to communicate). And context sources may dynamically join and leave the environment without notifying the agents. Therefore, it will be difficult for context-aware agents on a mobile device to effectively acquire contexts.

Privacy Issues - In a context-aware environment, agents collect and share user information, such that it raises great concern for user privacy. Privacy is intrinsically bound up with control - who controls what information. In the existing context-aware systems, users often do not have control over how personal information are acquired. As context sensors are built to be hidden in the physical space, personal information are collected without the explicit consent of the users. And privacy issues also arise when sharing user information.

To rise to the challenges, a suitable middle system is demanded. It's the target of our project-GaCam[5]. It will be detailed in following sections.

2. Context Aware Middleware

The role of middleware is to provide an additional layer of abstraction suitable for a specific type of applications.

Conventional middleware technologies have focused on masking out the problems of heterogeneity and distribution to facilitate the development of distributed systems. They allow the application developers to focus on application functionality rather than on dealing explicitly with distribution issues. Different middleware systems such as CORBA, DCOM and Java RMI have proved their suitability for standard client-server applications. However, it is believed that existing traditional middleware systems are not capable of providing adequate support for the mobile computing and dynamic context based environment, as well as IOT. There is a great demand for designing middleware systems that can support new requirements imposed by mobility, as well as context awareness.

GaCam, as a successor of LaMOC [2], is a middleware system to support the efficient development, automatic construction and deployment of physical and logical context aware applications, launched since 2009, at Institute of Computer Applications, East China Normal University (ICA-ECNU), supported by of the Shanghai Scientific Development Foundation (under Grant No. 09510703000 and No. 10dz1500103).

LaMOC is a location aware system [2]. It involves a complex context life cycle (context sensing, context pre-processing, reasoning and applications, etc.). . Where, sensors (including logical and physical sensors),to sense context are various contexts sensed are changed dynamically, corresponding pre-processing, as well as reasoning mechanism are also variable and changeable. Challenges appear, such as how to fit in with the variability and dynamically changeable, as well as hardly heterogeneous. A context aware middleware system as GaCam is urgently required, to satisfy the requirements such as physical/virtual sensor orientation, flexible context awareness and inference mechanism.

[5] GaCam-acronym of Generalized Adapt Context Aware Middleware.

Assumed computing environment in GaCam is nomadic (i.e. it runs on fixed network and mobile network background at same time), involving various and changeable computing power of nodes, dynamically changed topology of communication network, limited communication bandwidth for mobile nodes, etc.

In order to flexibly support context aware applications, dynamic reconfiguration, adaptivity, context awareness, reflection are as main features of GaCam.

Dynamic reconfiguration. During the system lifetime, the application behavior and context may be altered due to dynamic context changes. Dynamic reconfiguration is thus required. Dynamic changes in system behavior and operating context at runtime can trigger re-evaluation and reallocation of resources. Middleware supporting dynamic reconfiguration needs to detect changes in available resources and either reallocate resources, or notify the application to adapt to the changes.

Adaptivity is also a requirement that allows applications to run efficiently and predictably under a broader range of conditions. The middleware needs to monitor sensors/context such as the resource supply/demand, compute adaptation decisions, and notify applications about changes.

Context-awareness is important to build an effective and efficient adaptive system. For example, the context of a mobile unit is usually determined by its current location which, in turn, defines the environment where the computation associated with the unit is performed. The context may include device characteristics, user's activities, services, as well as other resources of the system. Especially, both physical and logical context awareness are focused here.

Reflection. A reflective system consists of two levels named as meta-level and base-level. The former performs computation on the objects residing in the lower levels. The latter performs computation on the application domain entities. The reflection approach supports the inspection and adaptation of the underlying implementation (the base-level) at run time. A reflective system provides a meta-object protocol (meta-interface) to define the services available at the meta-level. The meta-level can be accessed via reification, which expose some hidden aspect of the internal representation and hence they can be accessed by the application (the base-level). The implementation openness offers a straightforward mechanism to insert some behavior to monitor or alter the internal behavior of the platform. This enables the application to be in charge of inspecting and adapting the middleware behavior based on its own needs.

Besides, GaCam should also support: multi sensing orientation; flexible application composition; support for heterogeneity and scalability; support for privacy protection; traceability and control; as well as ease of deployment and configuration.

The framework of GaCam can be illustrated as in Figure 1. GaCam (in shadow) is in the middle, which can run on different operating systems, and support various context aware applications. It is divided in three layers, i.e. Middleware Administration, Middleware components and Application Constructor.

- Middleware Administration. It is used to manage middleware, monitor and audit the running of middleware.
- Middleware components. This layer has various context processing components covering the life cycle of context (Figure 2), such as sensing components, pre-processing components, reasoning components, and application components.

Application Constructor. The Application Constructor is in charge to construct various context aware applications, therefore composition and deployment mechanism as well as software appears here.

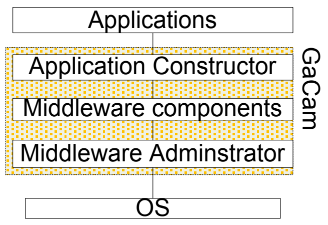

Figure 1. Layered GaCam Architecture

GaCam's support covers a life cycle of context as in Figure 2. The life cycle of context can be illustrated as an evolution from digital signal to knowledge. That is, via (physical/logical) sensors signal is sensed to raw context. After pre-processing of context, raw context is converted to context. Continuingly it is deduced to knowledge after reasoning, as well activation commands. The design of our middleware is around such a life cycle.

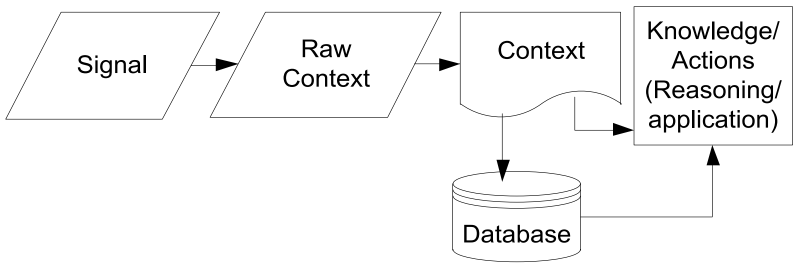

Figure 2. Context Life Cycle

GaCam supports flexible and adaptive sensing source and mechanism, compatible with physical and logical sensors. Sensed context is in raw data. For further usage, context pre-processing including context converting, filtering and fusion is required. Because of the changeability of sensors and sensed context, pre-processing mechanism (e.g. converting, filtering and fusion) should adapt the changes, dynamically. Some context should be stored for further uses. The system should satisfy the requirement of the situation, keeping the consistency of stored context, and efficiently store and manage global context database and dynamic context at mobile devices. Reasoning is an important step in context aware computing. A flexible reasoning mechanism is required, to suite the dynamical and adaptive application situation.

3. Sensor, Virtual Sensor and Meta Sensor

The spectrum of sensor involved here are wide. We category them into two classes, physical sensors and logical sensors, as in Figure 3.

The physical sensors can be defined as “A sensor is a device that measures a physical quantity and converts it into a signal which can be read by an observer or by an instrument. ...” (http://en.wikipedia.org/wiki/Sensor).

They can be further categorized in acoustic/sound/vibration sensors; automotive/transportation sensors; chemical sensors; …; and position/angle/displacement/speed/acceleration sensors (as shown in Figure 3).

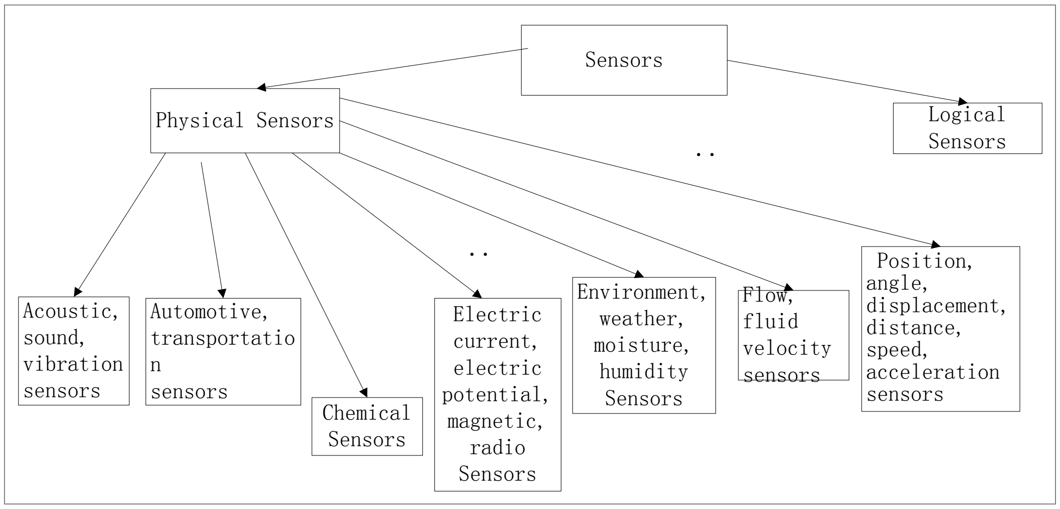
Sensors
Physical Sensors
Logical Sensors
..
..
Acoustic, sound, vibration sensors
Automotive, transportation sensors
Chemical Sensors
Electric current, electric potential, magnetic, radio Sensors
Environment, weather, moisture, humidity Sensors
Flow, fluid velocity sensors
Position, angle, displacement, distance, speed, acceleration sensors

Figure 3. Sensor Spectrum

Continually, as an example, the Position sensors can be classified in to GPS based, Phone-cell Based, RFID based, etc., as in Figure 4.

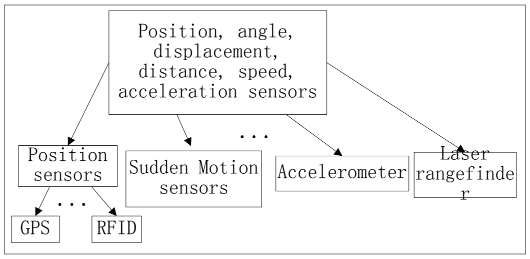

Figure 4. Where are Position Sensors

The logical sensors are software based, such that they can be classified as software agent based sensors, and semantics implied in data, as well as user interaction based sensors (Figure 5).

Context sensed by software agent are used more and more popular now. Spiders used in web searching, daemon in operating systems, as well as mail server daemon are such examples of software agent based logical sensors.

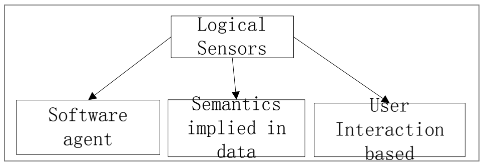

Figure 5. Logical Sensors

Semantics implied in data is another case. It means, user profile, group profile, computer profile, network profile and so on are sensed, and more implied semantics are as context mined.

The third of logical sensing is via user interaction. Based on user interaction, user intentions, usages and operational context are analyzed, and user behavious are collected and mined as context.

In GaCam, recommendation systems and web search systems are as the examples using logical sensors, where user profile, user behavious and usages are sensed and software agents (e.g. web spiders, crawlers, etc.) are used.

To suite the variety and heterogeneity of sensors, a reflective mechanism is designed in GaCam (Figure 6).

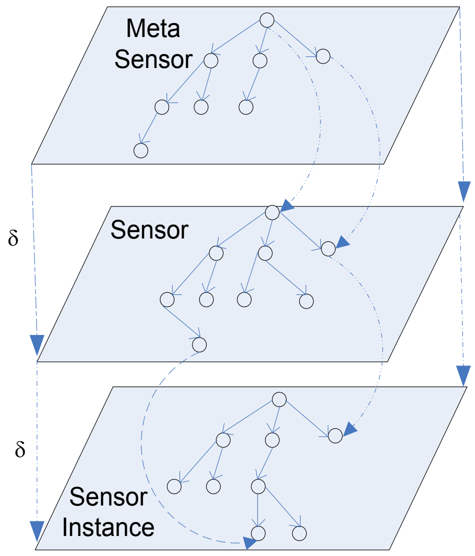

Figure 6 Meta Sensor, Sensor and Sensor Instance

Meta sensor is an abstraction of sensors, where the common properties of sensors are described. The relationship of sensor and meta sensor can be illustrated as in Figure 6, the mapping (noted as δ) from meta layer to sensor layer is named as reification. (The relationship between sensor to sensor instance is named as *instance-of*.)

4. Semantic Context Model

Dey, Christiansen and other scholars proposed different definition of context [1, 2]. But they mainly focus on the context sensed by physical sensors. Actually, besides context sensed by physical sensors (named as physical context here), logical context sensed by software is more and more widely used now. Therefore, in GaCam, the concept of context is widely extended as follows.

Context：*any information which can be sensed and used to characterize the situation of any entity, even real and virtual.*

It is emphasized that context is sensible, even by physical sensors or logical sensors, i.e. from real and virtual entities. The real entities are, for example, a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves. Virtual entities are computation power of devices, profiles, implied semantics in data, etc. Context sensed by physical sensors and virtual sensors is named as physical context and virtual context, respectively.

Therefore, challenges come：

- The model should adapt various sensors;
- Compatible with physical and logical sensors;
- Entail enough semantic information, especially, for later reasoning.

Some context models have been proposed, such as Key-Value pair Models; Markup Scheme Models, Graphical Models, Object Oriented Models, Logic Based Models, and Ontology Based Models [8].

Key-Value pair Models, Markup Scheme Models and Graphical Models are weak in semantic expression comparing with Object Oriented, Ontology Based and Logic Based Models. The later three models are also different. Object Oriented Model and Ontology Based Model are weaker than logical based model in reasoning. We need a reasonable context model, which is always available and useful during the life cycle of context. The context model should support enough semantic expression, flexible inference, and durable context storage, compatible with physical and virtual context. Therefore, combining with the advantages of last three models, a GaCam context model is proposed by us.

GaCam Context Model

As the description of information, context concept (noted as *ContextConcept* in following) is used to express context. Actual context is an instance of a context concept.

A context concept characterizes the common properties of a group of contexts. Actually, it descripts the common properties of contexts sensed by a sort of sensors. The properties are described by corresponding attributes.

ContextConcept

ContextConcept: = {attribute}

Besides, the dynamic behavious of contextConcept is described as methods. Therefore ContextConcept is expressed as:

```
ContextConcept
   {
   {attribute1;
    attribute2;
   …
   }
   Behavious
   { getContext();
   ConvertContext();
   FilterContext();
   FusionContext ();
   …
   }
}
```

Here, some generic behaviors are presented, such as getContext() used to read sensed context from sensor, ConvertContext(), FilterContext and FusionContext() used to pre-process context, i.e. context converting, filtering and fusion.

Attribute
Attribute: = <attributeName:type> | < attributeName: ContextConcept> | <attributeName: ↑ContextInstance>

That is, an attribute can be data in simple data type (e.g. real, or integer), an instance of a ContextConcept, or a pointer to an instance of a ContextConcept.

ContextInstance

Context Instance: a context instance is an instance of a ContextType, e.g. context sensed by GPS is an instance of ContextConcept GPS, i.e. MyGPS_Context is *Context Instance of* GPS.

The pre-defined relationships of context-to-context in GaCam context model are classified into: *is-a*, *has-a* (i.e. *component-of*), and *member-of*. Users can define other relationships.

As example, ContextConcept ***Position*** can be described by its longitude, latitude altitude, and some methods, as follows:

```
ContextConcept Position
{
   {
   Longitude: real;
   Latitude: real;
   Altitude: real
   }
   Behavious
   {
   …
   }
}
```

Furthermore, we can define GPS as:

```
ContextConcept GPS
{
   {CurrentPosition: Position;
    CurrentTime: Time
   }
   Behavious
   {
   …
   }
}
```

GPS *is-a* relates to ***Position***.

Such that, GaCam can be described as:

∃s∈sensor, such that

$$s \xrightarrow{getContext} raw\ context$$

Preprocessing

$$raw\ context \xrightarrow{convert/filtering} context$$

Fusion

$$context \xrightarrow{fusion} situation$$

Reasoning

$$context \xrightarrow{reasoning} \text{conclusion}$$

Reaction

$$conclusion \xrightarrow{driven} \text{actuator}$$

A context based query language and a corresponding interface, and a context definition tool are designed in GaCam.

5. Mobile Agent System

One of the main shortages of traditional software is lack of intelligence. Therefore, intelligent agents, especially mobile agents, are widely used in GaCam, to satisfy adaptivity and dynamic reconfiguration in physical and virtual context aware environment.

Here, an agent is a software component, which executes specific tasks on behalf of someone (a person) or something (an organization or another agent) with some autonomy. An agent is an active object (i.e., it can decide to execute a method) that can be implemented through one or more threads on one operating system process. Agents have some features, such as autonomous, proactive, self-learning, and sociality.

Agents are categorized to mobile and static agents. Static agents are created in the context of a specific application at the user’s initiative and become to the user for a long time. Mobile agents are created by static agents, by other mobile agents, by other types of objects, or even by humans. They navigate between different places in order to execute some predefined tasks near to desired resources. Mobile agents play main role in GaCam.

A group of agents having common properties is named as an agent type or agent cluster. An agent is an instance of some type.

A node in GaCam is a hardware infrastructure on which agents are executed. Typically, a node is a computer, PDA, a smart phone, an iPad or another computing device. Usually, an agent executes in one node. However, during its life cycle, it may navigate through an arbitrary number of nodes. An agent can be duplicated but each agent can only be executed in one node at a time.

GaCam is a mobile agent system. A system responsible for supporting and managing agents is called an Agent System, in short AS. One or more ASs can exist in each node.

An AS provides a full computational environment to execute an agent, as well as other support features such as agent persistence, security and mobility.

Users have agents that execute on their behalf. They can interact with their own agents, or can interact with agents owned by other users if their respective security managers grant that behavior. Users have rights over their agents, to suspend, change their knowledge and goals, or even eliminate them.

Agents are grouped in two groups here, i.e.:

- Context agents: agents responding to context processing during the context life cycle
- System agents: agents for agent monitoring, auditing and so on, named as administrative agent also.

The context agents are further classified in three types, i.e. functional agents, actuator agents, and sensing agents, respectively.

Sensing agents are used to sense context. Actuator agents are for reaction according to the context and context based inference. Functional agents are agents for pre-processing, context storing as well as context based inference.

For each agent, three interfaces are exposed (cf. Figure 7). That is,

- a functional interface to offer services,
- a state management interface to change the run-time internal status,
- and a property interface which provides an approach to achieve the reflectivity.

Through the state management interface, application developers can create, suspend, or kill a service provided by each agent.

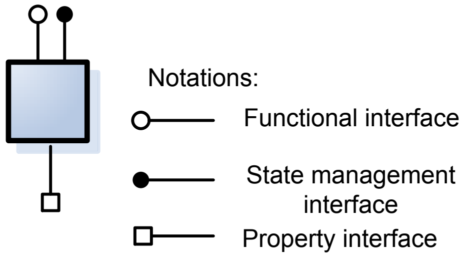

Figure 7. The General Structure of an Agent

The multi-agent framework in GaCam is shown in Figure 8.

The Sensing, Reasoning, and Context Storing Agent are in charge of sensing, reasoning, and storing context respectively. The Application Constructor composes existing services into user specified application systems in order to construct desired systems, flexibly. The State Query Agent offers a tool for application developers to monitor the state of each running agent, to be convenient for system development. The Directory Facilitator serves a directory to new agent discoveries. The Authority Management stores and manage the user-defined rules for authentications between agents and those between users and applications. Furthermore, it also exchange public key between communicating agents.

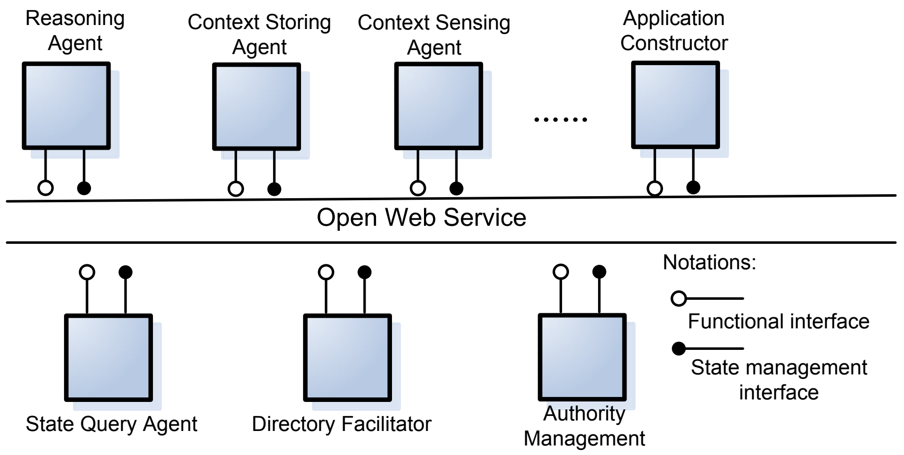

Figure 8. Structure of the Multi-agent Framework

The framework can release application developers from bothersome agent distributions, and facilitates the development of distributed applications. The incorporation of multi-agent technology makes our framework the ability of adaptivity, reflection and dynamic configuration.

6. Context Pre-processing

Context pre-processing is an important step in context life cycle. It's task is to transfer raw context to context as shown in Figure 2.

Data pre-processing is the first important step in data mining (knowledge discovery). With pre-processing, noise should be filtered out, and incomplete and inconsistency of data should be corrected during the data pre-processing. Here the pre-processing is named as ETL (Extract, transform and load), i.e.:

- Extracting data from outside sources
- Transforming it to fit operational needs (which can include quality levels)
- Loading it into the end target (database or data warehouse)

Different from traditional data pre-processing (ETL) where processed objects are semantics less, context pre-processing orients rich semantics of context. The implementation of pre-processing should consider the features of context. Therefore a reflection and multi-agent mechanism is designed by us here.

Context sensed by physical and logical sensors are in raw data. Raw data have their inherent shortages, such as information redundancy, containing huge incorrect and unsafe information, and so on. Context pre-processing should filter out such information at first. This step is named as context filtering.

Besides, sensed context may be in different format and following different measurement systems, e.g. sensed temperature by some sensors may be in Fahrenheit but others in Celsius, therefore context converting in context pre-processing is required.

Context fusion is also the task of pre-processing. For example, a fire alarm depends on the context sensed in temperature, smoky fog and (maybe) some other information. But individual context cannot trigger the fire alarm. For example, if only a high temperature sensed is because of other trigger, or only the smoky fog is because of cigarette smoking, the fire alarm will not be triggered. Only and only if fusing the context sensed by temperature sensor, context sensed by smoke sensor as well as (maybe) context sensed by other required sensors, and their values arrive the thresholds, then the fire alarm can be triggered.

Therefore, context filtering, converting and fusion are three important tasks in context pre-processing.

In the case of physical/logical sensor orientation and sensed context being various, quite different and changeable, a Reflection-based Multi-Agents Data Pre-processing Model (RMADPM) is designed in GaCam. Comparing with traditional data pre-processing models, RMADPM has some features, as follows:

1. Context oriented rather than data oriented.
2. The main management procedure is filtering/converting/fusion rather than ETL.
3. To adapt the difference of sensors and sensed various context, a reflection mechanism is used.

4. Implement framework is quite different from traditional data pre-processing. A multi-agent based interactive framework is designed to suite the flexibility and adaptivity.

The reflection mechanism can be illustrated as in Figure 9.

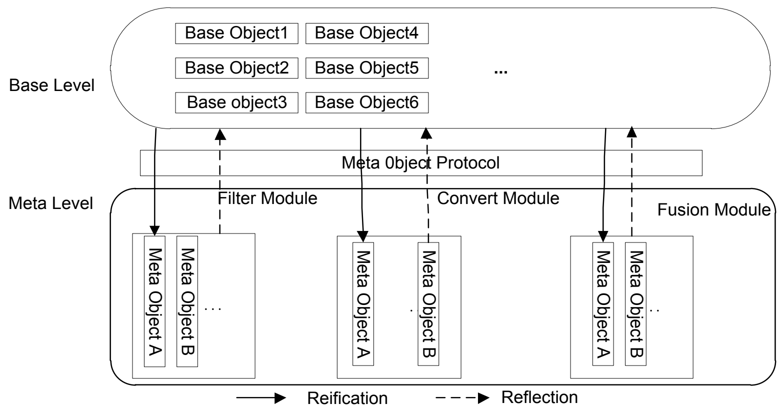

Figure 9. Reflection-based Multi-Agents Data Pre-processing Model (RMADPM)

As shown in the figure, there are Base-Level and Meta-Level two layers. The MOP (Meta Object Protocol) is the service interface of Meta-level. Via reification and reflection the information is mapped between two levels.

Besides, multi-agent mechanism for context pre-processing is designed here, as shown in Figure 10. Three functional types of agents are used here. Filter Agents in Filter Module are for context filtering. Convert Agents in Convert Module are in charge of context converting. And Fusion Agents in Fusion Module is designed for context fusion. A special Agent cluster, named as Monitor Agents, is in charge of monitoring, managing and auditing the functional agents and their works.

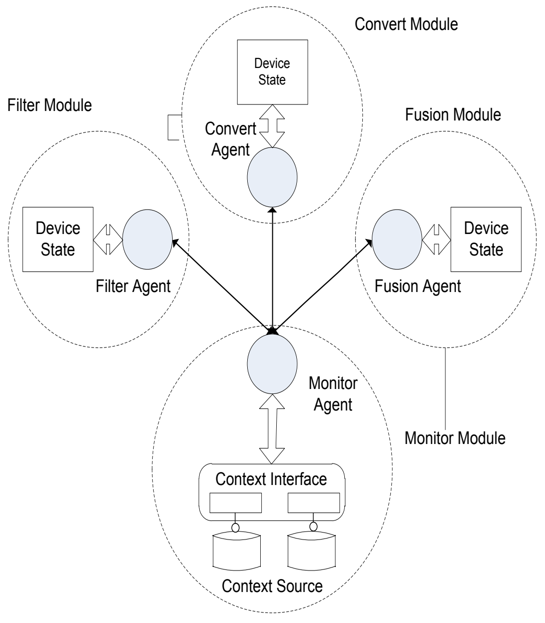

Figure 10. Pre-processing Agents

Relatively, the context fusion is more complicate than context filtering and converting, therefore it will be discussed deeply. The sensing by a sensor can be described as an atomic method. For example, Fgps() is an atomic method for GPS to sensing position. Ftemp() is a method to sensing temperature context. Fsmoke() is for smoke fog sensing.

Table 1 is an example of such atomic methods:

Table 1 Instances of Atomic Methods

Data ID	Atomic Method
GPS	Fgps()
Temperature	Ftemp()
Smoke	Fsmoke()
Light	Flight()
Noise	Fnoise()
Time	Ftime()
Weather	Fweather()
User	Fuser()
Interest	Fiterest()
YP_Info	Fypinfo()
…	…

In the context fusion, some sensing operations will be fused in some procedure. The fusion procedures are various, because of the differences in environments, usages, applications, etc. Sensing actions have to be fusing may be active in parallel, or may be in mutual exclusion. Therefore associated operators are proposed to express it, as shown in Table 2.

Table 2. Associated Operators

Operator	Definition	Description
$\wedge$	Parallel, conjunction	For $F_1, F_2, \ldots F_n$, $F_1 \wedge F_2 \wedge \ldots \wedge F_n$ means that they occur concurrently
$\rightarrow$	sequential	For $F_1, F_2, \ldots F_n$, $F_1 \rightarrow F_2 \rightarrow \ldots \rightarrow F_n$ means that they occur in a sequence of $F_1, F_2, \ldots F_n$
$\vee$	No sequence, disconjunction	For $F_1, F_2, \ldots F_n$, $F_1 \vee F_2 \vee \ldots \vee F_n$ means that they occur no sequence, and in disconjunction.
$\|\|$	mutual exclusion	For $F_1, F_2, \ldots F_n$, $F_1 \|\| (F_2, \ldots, F_n)$ means if F_1 occurs, then no any of $F_2, \ldots,$ or F_n occurs

For example, the fire alarm will be activated if the values of temperature and smoke fog have arrived determined thresholds. In this way, it can be described as:

(Ftemp() $\vee$ Fsmoke())

In LaMOC, location based recommendation is an important application [9]. There time and position are sensed by GPS sensor; yellow page information, user basic information and his/her favorite are sensed by soft sensors. In a recommendation application (e.g. to recommend a favor restaurant to a mobile user nearby just now) the context fusion can be described as:

((Ftime() $\wedge$ Fgps()) $\rightarrow$ Fypinfo()) $\vee$ (Fubasic() $\rightarrow$ Finterest())

It means, when time and position contexts are sensed at same time, the system can recommend favor restaurant(s) nearby to a mobile user according the yellow page information obtained subsequently. Or according to user's basic information read and user's favorites subsequently found following the basic information, suitable restaurant(s) will be recommended.

7. Open Web Service

Now applications face a more open environment, especially Internet based. Therefore Web Service[6] (WS) seems as more suitable than CORBA, DCOM, as well as RMI. But, Web Service provides loose coupling for distributed software components, rather close coupling as in CORBA, DCOM and RMI. Furthermore, comparing with RPC, IIOP and RMI, Web Service is stateless and lack of synchronized communication support. When a service requestor finds a service and binds the service, there is no working state exchange between the service requestor and service provider during the activation of the service. That is it's a stateless protocol. Accompanying with it, synchronized communication between the service requestor and service provider is also failed in WS. Context aware applications require the synchronized communication and state exchange between requestor and provider. Besides, lack of privacy protection is also its weakness. Therefore, an extended Web Service mechanism is proposed by us, which is named as OpenWebService (OpenWS) (as a softbus shown in Figure 8).

Beside supporting state exchange and synchronized communication, OpenWS is:

1. Enabling distributed context-aware agents (possibly running on resource-limited mobile devices) to contribute to and access a shared model of the context.
2. Allowing users to control the access of their personal information in a context-aware environment.
3. The core of the OpenWS is autonomous agents that manages and controls the context model of a specific domain.
4. Maintaining the context model of the domain, including domain contexts from the past and at the present.
5. Resolving inconsistancies and ambiguities of the domain contexts through information fusion.
6. Establishing privacy policies with users before sharing their personal information.
7. Providing knowledge sharing service for context-aware agents through agent communications

Within OpenWS, an agent is looked as a black box, with service as its interface. The Web service is used for such interface. Beside Web service, in some cases, especially for physical sensing, the service as defined in OSGi[7] is also used as its interface.

8. Other Issues

6 http://www.w3.org/2002/ws/

7 Open Service Gateway Initiative, http://www.osgi.org

There are other issues, for example, Inference Engine, Application Composition and Privacy Protection, etc. But they will be shortly described here, restricted by the length of the paper.

Inference Engine

In the life cycle of context, reasoning is an important step. Because facing variable sensors/context, inferential mechanism is adaptive to activate various actuators. Therefore an Inference Engine including a management system of variable methods (algorithms) is designed and developed in GaCam.

Application Composition

An application supporting platform is required for a middleware system. It should support application composition and reconfiguration based on the developed middleware. Applications will be constructed and re-constructed according to the change of context. A workflow based application composition subsystem is designed in GaCam.

Privacy Protection

As described, privacy is an important issue. A role based and multiple layered authority allocation and management mechanism is designed in GaCam. And in OpenWS, SOAP[8] is extended with authority assignment and identification, i.e. using the message with SOAP, the corresponding authority information is exchanged with an extended message envelop.

9. Conclusion

In this paper, context aware middleware is discussed. Here nodes are extended to sensors (even physical or logical). Nodes are various, changeable, and mobile. Therefore, the middleware should be more intelligent, adaptive, reflective and re-configurable. Furthermore the middleware should support the life cycle of context, and application composition automatically.

From our practice, it's found:

The middleware should be reasonably layered. If layered granule is very fine, then tasks in each layer are more concrete and detailed. Portability is its strong points. But the system will be more complex. If the system divided in few layers, the system architecture seems as more succinct. But the portability is its weakness.

From our experience, layered according the life cycle of context is reasonable.

Beside the layered architecture, designing a semantic context model and sensor model are also important and fundamental. A reflective mechanism is more suitable.

References

[1] Anind K. Dey, Providing Architectural Support for Building Context-Aware Applications, PhD thesis, Georgia Institute of Technology, **(2000)**.

[2] Matthias Baldauf, A survey on context-aware systems, Int. J. Ad Hoc and Ubiquitous Computing, vol. 2, no. 4, **(2007)**.

[3] Junzhong Gu, Context aware computing, Journal of East China Normal University (natural science), no. 6, **(2009)**, in Chinese.

8 Simple Object Access Protocol, used in Web Service

[4] Junzhong GU, Liang HE, Jin Yang, LaMOC –A Location Aware Mobile Cooperative System, in the Proceedings of Signal Processing, Image Processing and Pattern Recognition, **(2009)** December 10-12, Jeju Island, Korea.

[5] Kristian Ellebæk Kjær, "A Survey of Context-Aware Middleware", http://www.hydramiddleware.eu/hydra_papers/A_Survey_of_Context-aware_Middleware.pdf

[6] Harry Chen, An Intelligent Broker Architecture for Context-Aware Systems，PhD Thesis, University of Maryland, **(2003)**.

[7] Licai Capra, Wolfgang Emmerich, Cecilia Mascolo, Middleware for Mobile Computing (A Survey), Advanced Lectures in Networking. Editors E. Gregori, G. Anastasi, S. Basagni, Springer LNCS 2497, **(2002)**.

[8] Thomas Strang, Claudia Linnhoff-Popier, A Context Modelling Survey, Workshop on Advanced Context Modelling Reasoning and Management as part of UbiComp, **(2004)**.

[9] Rong Tan, Rong; Jun-zhong Gu, Jing Yang, Xin Lin, Peng Chen, Zhe-Feng Qiao, Designs of privacy protection in location-aware mobile social networking applications, Journal of Software, SUPPL. 1, 21, **(2010)**.

Authors

Junzhong GU

Professor of Computer Science, Head of Institute of Computer Applications, East China Normal University, China. He received the M.S. degree in Computer Science from East China Normal University in 1982. He works at East China Normal University since 1982. He worked as visit professor at GMD and University Mannheim, Germany (1987-1989, and 1991-1993). His research interests now include context aware computing, distributed data management, and multimedia information processing.

Gong-Chao Chen

Phd Student at School of Information Security Engineering, Shanghai Jiao Tong University, Shanghai, China. He received the Bachelor Degree in Electronic Technology from School of Electronic & Information Engineering, Shanghai Jiao Tong University in 1991. His research interests now include internet security and cloud computing.