Implementation of Context-Aware Middleware for Sensor Network in Ubiquitous Environment

Bo-Seong Kim, Byoung-Hoon Lee, Jai-Hoon Kim*
Graduate School of Information and Communication
Ajou University, Suwon, Korea

ABSTRACT

Recently many researches are investigating for ubiquitous computing and network. In the real world many sensor devices must be equipped to provide many services for users. To make computing environment easy and more user friendly, middleware system not only hides all complexities (network, system, services, etc), but also needs to have efficient context inference scheme and system reconfiguration capability. In this paper we suggest context-aware middleware design for sensor network which provides efficient computing environment for end-users. We also present XML based implementation of our system.

***Keywords**: Context Aware Middleware, Ubiquitous, Sensor Network.*

1. INTRODUCTION

'Ubiquitous' is the most frequently used term to describe near future briefly. Nowadays, some companies use 'ubiquitous' or similar term in TV or newspaper to advertise their intelligent apartment or products. In general perspective, however, their apartment or products are very dependent to specific environment (i.e., small area, homogeneous communication protocols/channels, and so on). We can conclude that all of these are the only domain specific applications using sensor devices equipped with RF or other wireless mediums. As you know, realization of ubiquitous environment must be based on general infrastructure integrating heterogeneous networks and systems. After establishment of the any solution for integration of such heterogeneous networks and systems have done, any domain specific applications need to be deployable. However, this is not an easy work. For example, assume that we have established infrastructure for USN (ubiquitous sensor network), ad hoc network and the Internet, due to the large number of sensors and mobile nodes which are scattered or embedded on streets, houses and buildings, too much sensor data traffic will be present in ubiquitous environment. In this circumstance, making domain specific applications is too exhaustive process without the aid of any network or system infrastructure. Domain specific applications does not concern about sensor data movement, adjusting network topology, finding location of peripheral service and evaluation/reconfiguration of current system status. To realize ubiquitous computing environment, system must wrap all of those contexts from user [2]. The better solution is to provide intelligent and general middleware which can operate autonomously according to user's preference or current context status or prediction of future context change. Some middlewares were developed to enhance the construction of adaptive and context-aware application [14] and support automatic configuration and dynamic resource management in distributed heterogeneous environments[15].

However, it is difficult to wrap diverse context from users and infer appropriate behaviors of the system. First, system behavior inferred from context can be categorized roughly as follows: (1) to deliver destined or suitable service to user (2) to overcome unreliable network condition via local caching scheme or other schemes (3) to resolve resource insufficient problem, migration would be used for load balancing. (i.e., energy, memory, CPU capability, disk availability, and so on) To achieve (1), (2) and (3), middleware need to be designed carefully to resolve some mobile computing problems caused in 'Sensor Network', 'Ad-hoc Network', 'Service Discovery' and 'Mobile Code' issue.

Our purpose is to design the context-aware middleware focused on general problems listed above. Since we are not focused on specific purpose such as voice recognition, wearable computing, health care, and so on, core of context-aware middleware have objective to provide flexible and expandable feature for general use. For user using our middleware, it provides user (domain specific applications) with the automatic behaviors of system/network according to user preference and current context status while letting the upper layer (application domain) takes application domain specific functionality. This concept is comparable to deployment of IPv6's secure module which is not the original component of IPv6. That is, IPv6 provides upper layer (i.e., SMTP, HTTP) with core functionality of network layer while

This study was supported by a grant of Frontier 21 Project of Republic of Korea.

letting the secure module as a separate part of IPv6 itself (Usually IPv6 recommend that the 'IPSec' could be used to support security of IPv6 core).

To meet the ubiquitous era, many researches are going on their way and many ubiquitous projects are in there own way. Following are the well known projects in ubiquitous research area.

- Cooltown – HP [8]
- Intelligent Room – MIT [4]
- Interactive Workspaces Project – Stanford Univ [9]
- The Aura Project – CMU [5]
- Easy Living Project – Microsoft [10]

We think that UI (User Interface) could be the most important factor to win in the commercial application market area. Sensor devices need to support way of delivering their raw data to target system. In ad hoc research area, various routing protocols are emerging and evolving, basically those are based on traditional routing protocols. Since as you know, sensor devices are very domain specific and data expression is variant. In ideal situation, to be compatible with all of the various system, one easiest way is for sensor manufacturer to let the device driver or software embedded into sensor device follow the standard or de-facto raw data description rule (i.e., by using XML). In this paper, to differentiate application layer module (i.e., Service Discovery Description or User Policy) and low layer module (e.g., Sensor Raw Data, System/Network Resources) from core of middleware, in the context of independency of data exchange, we used XML for data description. XML is used widely as a system independent language for data-description, process-description and so on [11].

The next section describes about service categorization and composite context. Section 3 look at the system overview, section 4 describes implementation details and, finally, section 5 presents our conclusion.

2. SERVICE UNIT CATEGORIZATION AND COMPOSITE CONTEXT

2.1 Service Categorization

From the perspective of service provision, we can think user as a person who wants to be served seamlessly unaware about system configuration. So middleware must conceive everything about the contexts which are concerned with user preference. For doing this, middleware must maintain all of current context and has the ability to communicate with other module. There are many possible ways to deliver a particular service to user. For example, if the current network context is so unreliable, middleware can deploy a caching module for next network connection. It is convenient to think caching module as a CODA file system. In this scenario it is also good to provide mechanism for existing connected user to switch to local-caching mode while still maintaining its connection(s). Surely, other service delivery scenarios which are concerned with not only network, but also system are possible. As you can see from above example, user did not need to recognize internal contexts concerned with problems of unreliable network status. In this case, context-aware middleware recognizes current system status and infers best suitable behavior based on current context.

After this inference process, context-aware middleware tries to reconfigure current internal system or service delivery way properly according to the inference result. We categorized unit for service reconfiguration as below, and these units could be look like an entity for service provider [2]. (In this categorization, we consider only general services.)

(1) *Services which is provided in 'Application Domain'*

(2) *Service which is provided in 'Middleware'*

The former includes all of the application services such as Video Streaming, Print, FTP, SSH, DNS and so on. In fact these services are internally registered to middleware. Usually the provider of service might supply more than one service implementation. For example, assume that 'Video Streaming' service application server provides four service implementations. First implementation is designed for TCP suitable channel and client which has strong capability in decoding, second is designed for TCP suitable channel and client which has weak capability in decoding. Similarly, Third implementation is designed for UDP suitable channel and client which has strong capability in decoding, and fourth is designed for UDP suitable channel and client which has weak capability in decoding. As you can see, many varieties for delivering 'Streaming Service' are possible.

The latter includes general services categorized as a level of inherent and dependent characteristic of system. Caching service also can be implemented as an 'Application Domain' service, but it is more efficient and feasible to implement caching in middleware layer as a middleware service. These middleware dependent services could be 'direct network connection/local caching', 'direct disk reference/use memory buffer', 'local execution/code migration', 'write back in software level for power saving by buffering/immediate write', and so on. Figure 1 shows categorization of reconfigurable unit for general purpose middleware.

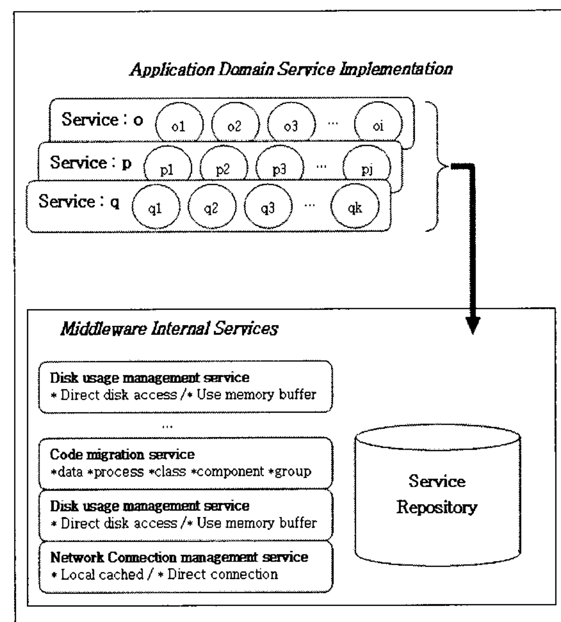

Fig. 1. Categorization of service entity which could be a unit of reconfiguration.

2.2 Composite Context

In previous section, we described the relation between user and service delivery method. In addition to this relation, user

can set his personal policy by describing his preference about interesting service with the composite context condition. The context-aware middleware is originally designed for the need of proactive fashion operation. That is, computing environment of which user is not concerned is desirable. However, user cannot be entirely excluded from the system for their convenience. User would want to be delivered 'S3' service from service pool when particular condition is satisfied. For example, if currently service S which is implemented as 'n' number of different types and conforming service implementation pool (i.e., {S1, S2, S3, … , Sn}) is registered in middleware. At this time, suppose that according to the user's policy description that is written in XML, user wanted to execute S3 service where *"((MEMORY_available = 128MB AND CPU_loading < 50%) OR (MEMORY_available = 64MB AND CPU_loading < 10%)) AND (NIC_status = ON)"*. To reflect user's preference or his own rule, each user generally describes his policy and passes it to the middleware [3]. Above notation is the form of composite context. It is similar to Boolean computation model. Each context (i.e., MEMORY_available, CPU_loading, NIC_status, and so on.) is composed to express one condition. We will call such a set of context information as a 'Composite Context'. Figure 2 shows an example XML file for policy and user profile.

```
<?xml version="1.0" ?>
- <ContextXML session="123">
  - <User ID="1" Name="Boseong Kim" Role="Researher" Address=""
      City="Suwon" Country="Korea" PostalCode="" Tel="81-31-219-2443" Fax=""
      Email="winnerxg@dmc.ajou.ac.kr">
    - <Policy ID="1">
      - <CompositeContext Name="LowBandwidth">
          <Context ID="7" Name="NetworkMaxRate" Type="Computing"
            Unit="bits/s" Value="4500" Time="10:33:42"
            TypicalValue="7000" LowerBound="5000"
            UpperBound="10000" />
        </CompositeContext>
        <Service ID="1" Name="AccessData" Type="" ImplementationID="2"
          ImplementationName="Cache" />
      </Policy>
    - <Policy ID="2">
      - <CompositeContext Name="HighBandwidth">
          <Context ID="6" Name="NetworkDisconnect" Type="Computing"
            Unit="no/ms" Value="0" Time="20:45:12" TypicalValue="0"
            LowerBound="0" UpperBound="2" />
          <AND />
          <BEGIN />
          <Context ID="7" Name="NetworkMaxRate" Type="Computing"
            Unit="bits/s" Value="20000" Time="20:45:12"
            TypicalValue="7000" LowerBound="5000"
            UpperBound="10000" />
          <OR />
          <Context ID="8" Name="NetworkDelay" Type="Computing"
            Unit="ms" Value="90" Time="20:45:13" TypicalValue="125"
            LowerBound="100" UpperBound="155" />
          <END />
        </CompositeContext>
        <Service ID="1" Name="AccessData" Type="" ImplementationID="1"
          ImplementationName="LiveConnection" />
      </Policy>
    </User>
</ContextXML>
```

Fig. 2. Sample XML file for representation of user's information and policies. As you can see processing of 'Composite Context' is the key feature of inference engine [13].

While we are implementing the context-aware middleware core, to process composite context description written in XML, we tried to use pre-existing XML parser. But due to the lack of processing capability of XML parser for Boolean expression while obeying regulation of DTD (Document Type Definition) or XML schema notation, we could not process composite context. (There was *MathML* or something like that having ability to process Boolean model or more complex numeric expression markup language.) We determined to parse one more time with the help of original XML parser. In this case, XML parser acts like a token analyzer. With the use of general paring method of BNF, we could implement a composite context processor. This composite context processor parses the input XML file (written user policy and profile), and then generates tree and chain data structure. We called this as an 'evaluation chain/tree', because once middleware detected change in context status, start checking for the dependency concerned with this context. Then evaluation is performed using 'evaluation chain/tree'. Finally, middleware perform reconfiguration work

3. ENTIRE SYSTEM OVERVIEW

Our context-aware middleware has two layers. We called the upper layer as a 'Service Layer' and the lower layer as a 'Context Configuration Layer'. As you can imagine, context configuration layer is responsible for the gathering context information. Abstract design explained in this paper is an output of our project [13].

3.1. Context Configuration Layer

We define some termination for specific explanation of design. Let's begin with context configuration layer first. This low layer has two important modules namely CAM (Context Acquisition Module) and CMA (Context Monitoring Agent). CAM is responsible for gathering, processing and interpreting the users' contextual information. It is a 'context widget' that contains CMAs. Each agent is associated with *one* type of contextual information (like location, bandwidth, battery, etc.). Each agent has a state, consists of a set of attributes (variables) and a behavior (i.e., a set of call back functions that are automatically triggered by context changes). CAM has the following key responsibilities:

a. To gather context data provided by software services and/or hardware sensors.
b. To interpret gathered raw sensor data into meaningful information (like light, battery life, location, bandwidth, etc.).
c. Access the CPDB (Context and Policies Database) with the information derived in step (b).

First, we designed that each sensor devices or system resources are monitored by CMA (Context Monitoring Agent) and then CMA converts the raw data into the meaningful information using XML expression. Then CMA updates this XML context information into the CPDB (Context & Policy Database). CPDB is a kind of database which acts as a bridge repository for the contextual information and policies (how the middleware has to behave when executing in particular contexts) for all the users. *Policies* are applied for delivering services depending upon context configuration. The major responsibilities of CPDB are:

- It receives the specifications of the contextual information for each user from the agents (CMAs) of the CAM.

- These specifications are translated into XML documents for subsequent processing by the SSM (Service Selection Module) of the *Services Layer*.

3.2. Service Layer

Prior to going further details, let's remind the role of context-aware middleware. The purpose of our middleware is to provide appropriate service to user by inferring best behavior based on current contexts and user policy [7]. To achieve this, especially the inference engine of middleware must have the ability to select the best suitable behavior. But, it is not a simple job to do this. The solution for the selection of the best behavior might be possible by adopting AI (Artificial Intelligence) engine. We leave this as a future work and we decided to design simple inference engine. As a result, now you might noticed that the previously exploited composite context give limitation to the ability of inference engine. Due to the Boolean model based policy description, our inference engine's ability is limited. We plan to enhance inference engine in later.

First of all, you must consider that middleware's service layer interacts with application domain. As a result, layered approach for communication is so important. Therefore we decide not to use language dependent procedure call. This is naturally understood well by referring previous fact that composite context or user information is described in XML. Through the whole design part, we do not reveal language dependent primitive or exposed API outside.

Service layer is composed of two modules, SSM (Service Selection Module) and SEM (Service Execution Module). SSM component is the heart of 'Context Aware Module' that performs the core functionalities. It contains SDA (Service Discovery Agent) that matches and recommends appropriate service delivery method from a pool of services after taking into account the following:

- Contextual information of the user: This information is fetched from the CPDB. Context can be *composite* (i.e., comprising many contexts like current context, user's device capabilities and constraints set by the 'Service Providers'). There are situations where complex event compositions are required by SSM to monitor composite events occurring across various environmental contexts.
- Policies defined by the user: They are also fetched from CPDB.

In service layer of context-aware middleware, SDA module could not be easily built due to the lack of exact standard or specification about service discovery on sensor network or Ad hoc network. Although in Ad hoc network area, on-demand based service discovery protocol was proposed [6]. However, this is not yet implemented and URL based service description is not sufficient to reflect exact user's preference. In our previous work, we had implemented service discovery protocol for ad-hoc network [12]. In [12], we also had some question about the URL based service description [6]. So, when we were engaged in service discovery work, we focused on finding the method to reflect the user's preference. Thus we proposed new service description method which is suitable for reflecting user preference. Main idea of [12] against service description issue is that service description must contain the user's preference while letting the user is free from any hierarchical constrains or any rules. To give the freedom of description, [12] devised property based 'weighted-vector service description'. It is similar previous policy description in composite context XML file. For example, assume that service demander (client or user) want to find 'printer service', namely K, whose property is 'paper=A4, color=true, resolution>300dpi'. SREQ (Service Request) packet will contain the property information concerned with request. After routing through the whole or some node, this SREQ packet will be examined against resources status of current node. If current node has a candidate service provider, this node will reply as SREP (Service Reply). In this *property based service description* could be extended to including the *weighted value* against each property. (i.e., $\sum$\{paper=A4*[0.6], color=true*[0.1], resolution> 300dpi*[0.3]\} $< p$, where p is proximity)

If SDA adopted our *property based weighted vector* protocol, you can easily make a relation with the contents of composite context based XML file. Actually, user's policy is similar to user's preference and composite-context description is similar to the way of [12]'s *property based weighted vector* description protocol. So we could easily implement the SDA module. Also it was proved that *property based weighted vector protocol* minimizes control packets such as RREQ, RREP and RERR. In Furthermore, *property based weight vector protocol* does not perform string matching processing. As a result, we can save computing power and get a fast response time.

SEM module carries out the execution of the service. There are two types of execution:

a. In case of services residing on the device, execution is always local to the device.
b. For remote services, execution can be local or remote. In the remote invocation, the client remotely sends a request to a provider asking the execution of a service. The execution takes place at the provider platform. In local invocation the client remotely asks to transfer a copy of the service. After being transferred, the execution takes place at the client side.

3.3 Sequence Flow and Use case

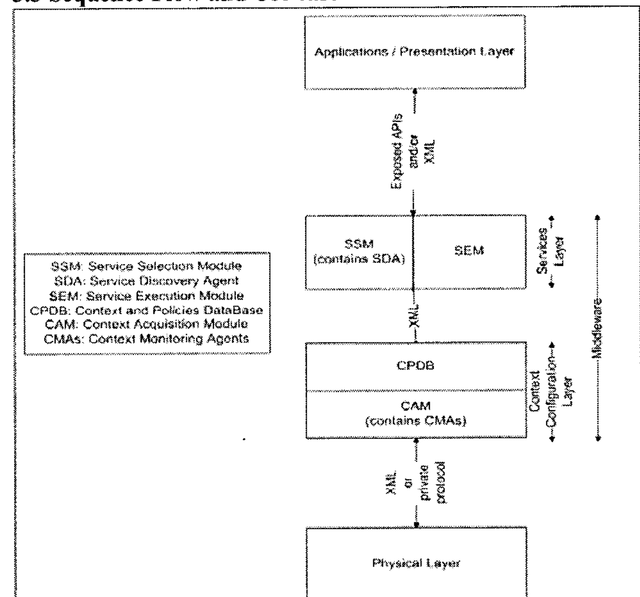

Fig. 3. Block diagram of Context-Aware Middleware [13]. In implementation, middleware system does not use language dependent APIs.

Figure-3 shows layered structure of middleware and the interactions between each component. General execution

sequence of middleware is as follows:

1. Each CMA of CAM detects change in context and callback function is triggered
2. user requests a service (through exposed APIs)
3. Service layer access CPDB & selects suitable service
   → SDA of SSM consults service registration information in CPDB, then selects the appropriate service delivery method by executing inference engine.
4. execute selected service

Figure 4 shows execution sequence.

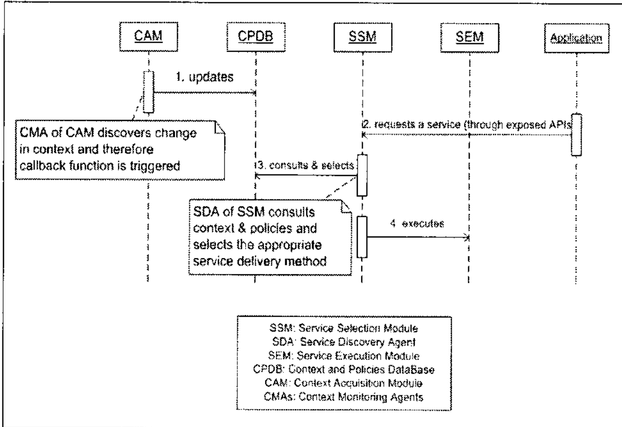

Fig. 4. Sequence diagram of context-aware middleware [13].

3.4. Implementation of CPDB

In the perspective of implementation of CPDB (Context & Policy Database), a number of database designs are possible using relation database.

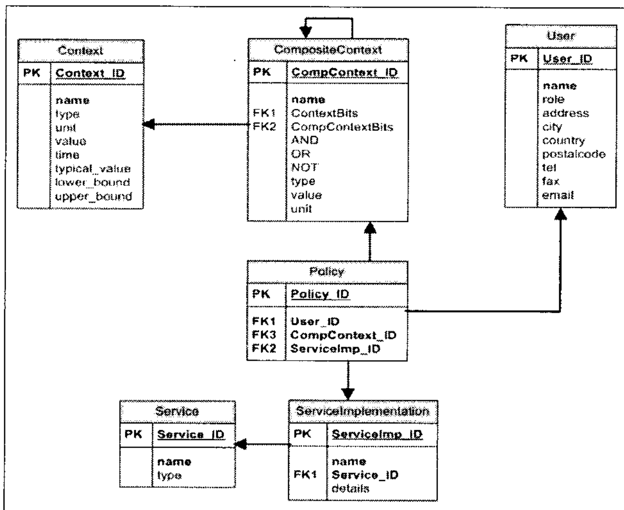

Fig. 5. Database design for CPDB in Relational-DataBase [13].

Assume that current node adopted RDBMS system as a middleware database system. In our opinion sensor node or Ad hoc node cannot afford to maintain DBMS. Thus, simple data structure is appropriate for sensor node instead of heavy RDBMS. In this case, we should design the DB table relation. In Figure 5, as you can see 'policy' table is in the center of the DB. Compare closely 'Service' table and 'ServiceImplementation' table each other. You could detect the fact that one to 'N' mapping between these two tables exists. Actually in our implementation of middleware, 'User' table is in the center of DB design because implementation could be better easier by deploying 'User' as a main object.

Although, in this paper we are using CPDB only to refer database for context and policy information only, in real implementation CPDB module contains not only context and policy information but also service information too.

4. IMPLEMENTATION DETAILS

Until now, abstract design of context-aware middleware was presented. From now on, some implementation concerned explanation will be followed. Middleware was implemented using Java programming language. The purpose of implementation work is to use it as a base framework for future component based context-aware middleware. Following sections explain functionality and relations of each module.

4.1 Low Level Resource Monitoring

We used virtual device driver approach for general purpose. There are many resources in present, but near the future new many resources will emerge. Therefore, middleware must consider extension of system. In the view point of middleware low level, to include all of the drivers in the middleware system in built time is not a good idea. When middleware was deployed once, inserting a new device driver requires rebuilding of entire system. To avoid this drawback, we used virtual device driver approach. In the concept of virtual device driver, all of the device dependent functions are implemented as an OS dependent device driver and this OS dependent device driver communicates with virtual device driver which is an element of middleware system. Using virtual device driver approach makes view of each resource be conceptual, and XML communication/generation matters be simple.

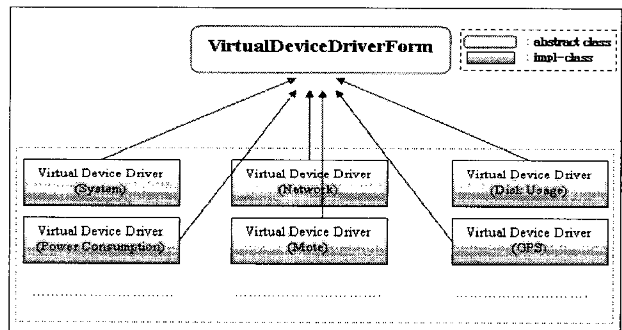

Fig. 6. Implementation hierarchy of 'virtual device driver'.

4.2 Framework for Management of Virtual Device Drivers

Each monitoring software module consists of OS dependent device driver and neutral type of virtual device drivers. Also, each monitoring module could run as a single thread or process. We used thread approach to implement resource monitoring software.

To manage resource monitoring software through the single way, we adopted Java singleton programming model. And, we called this singleton as a 'framework for management of virtual device driver'. In the step of initialization of framework, default virtual device drivers are created and initialized to be ready to run. This framework provides a GUI for user to control the behavior of each thread of virtual device driver. There are

two control options. (i.e., attach/detach) By initiation of 'attach' command, virtual device driver thread will execute its own function. Vice verse, 'detach' command will make thread to suspend. When we need to turn on or off the monitoring functionality of real device driver, we must have a way to manage the execution of multiple threads. To resolve this problem, we let the virtual device driver handle GUI event. That is, virtual device driver is registered as an event listener of GUI component. As a result GUI message generated by user is delivered to virtual device driver. Finally, virtual device driver requests framework to make thread be suspended. To provide a common way of attach and detach operations, framework implemented interface 'CMA Controlable'.

Due to the consistency problem between multiple threads, framework was implemented to use critical section locking method by 'synchronized' keyword. This critical section is applied to the 'thread control code' and 'context database access code'.

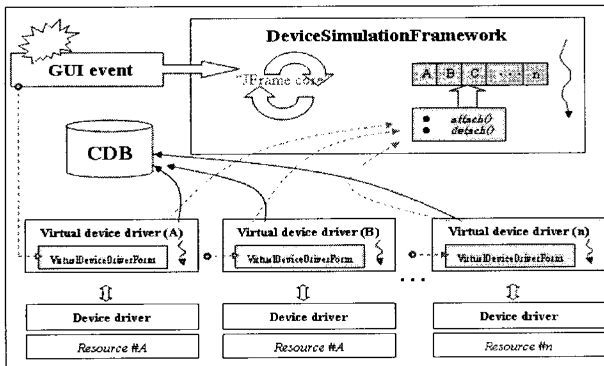

Fig. 7. Framework for virtual device driver.

We tried to include GUI for easy configuration environment.

4.3. Database for Middleware (SDB, PDB and CDB)

Explanatory description does not present about database. As mentioned in section 3.4, our middleware use three repositories for service, policy and context. Each of which are SDB, PDB and CDB class and has general database operations such as ADD, DELETE, UPDATE, RETRIEVE and so on.

4.4 Context Acquisition Module (CAM)

CAM plays a role of 'module management core'. Our context-aware middleware runs as a single process. Actually, a number of threads are created from main thread and each spawned thread performs particular work such as resource monitoring, accepting user request and management of each modules. In our implementation, Context-Acquisition Module works as a main thread. Therefore CAM creates Context-Monitoring Agent (CMA) which is explained previously (refer '4.1. Low Level Resource Monitoring' for more information.), and service concerned modules which will be covered later. Also, CAM creates database for policy, service and context.

In figure 8, waved arrow represents execution of single thread. Policy, service and context processor use our extended XML parser commonly, but each processor uses different database module (i.e., PDB, SDB and CDB). Service module is composed of two sub modules (i.e., external and internal service module). External service module deals with services provided by external service provider and internal service module deals with reconfiguration of network and system resources. That is, service module implemented according to design specification of figure 1.

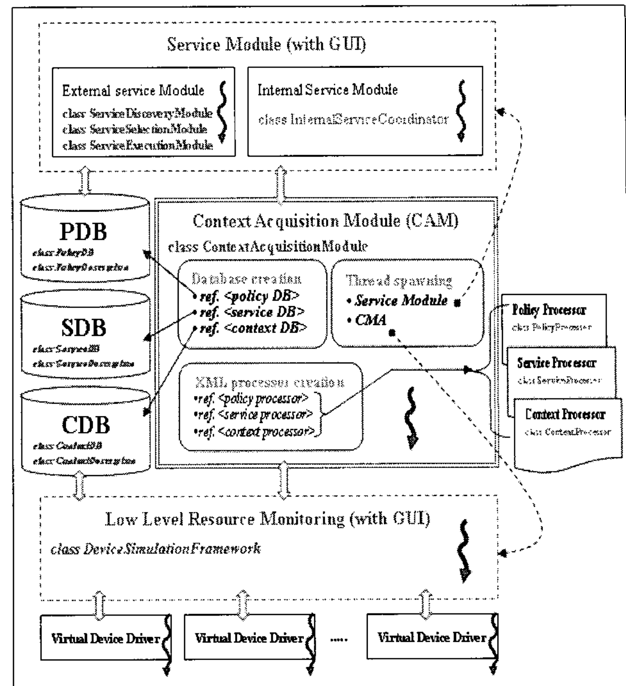

Fig. 8. Overall implementation diagram with CAM in center.

4.5 Inference Engine Considering Boolean Model

As we mentioned previous section, user policy and service provider's constraints are described in the XML document using composite context expression which is based on Boolean model. This policy and constraints are stored in the PDB and middleware will refer PDB whenever service needs to be delivered. At this time middleware determines proper service delivery method based on policy and constraints information. Determination or inference process contains evaluation of each policy.

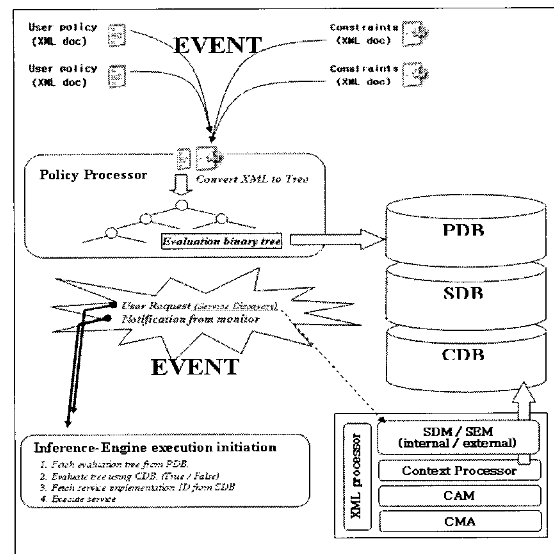

Fig. 9. Process diagram of inference engine.

To speed up evaluation process, we used tree based chain. It is created whenever new policy or profile or constraints were given and called 'evaluation chain/tree'. Time complexity of this policy decision is $O(lg(n))$ where n is length of composite context if decision tree is well balanced.

5. APPLICATION

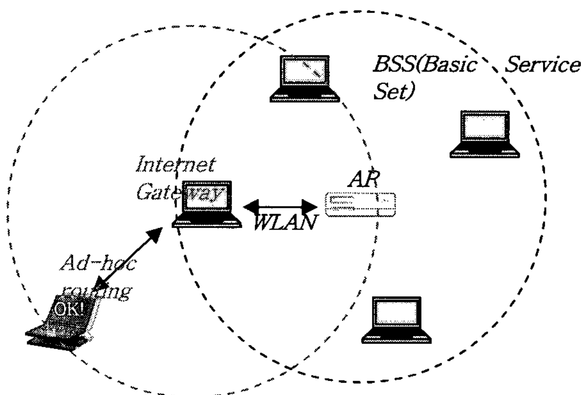

Fig. 10. Reconfiguration of network topology

To verify usage and application of our context-aware middleware, we implement and test context-aware software on our middleware platform. As a demonstration of our context-aware middleware, we implement reconfiguration of network topology. The reconfiguration is performed according to network context-aware captured by our middleware. In the demonstration, mobile host can choose appropriate network protocol between WLAN and ad-hoc. When a mobile host is in the range of AP (access point), it selects high performance WLAN while choosing ad-hoc at the outside of AP. In the ad-hoc protocol, other mobile hosts act as a gateway.

6. CONCLUSION AND FUTURE WORK

In this paper we have exploited context-aware middleware design and its implementation issues together. Our context-aware middleware was also designed to provide user with easy way of service using rather than overcoming resource restrictions by itself. The most valuable factor of our context-aware middleware is always recommending the suitable behavior for the user by referring current context/policy. In fact, our context-aware middleware can be upgraded by inserting AI inference engine. Furthermore, because any sensor or handheld devices always treated as a concept of context by middleware core, middleware does not reveal any dependency against every device.

We are going to attach more intelligent inference engine. Our long term final goal is to design context-awareness for community computing.

REFERENCE

[1] I.F. Akyildiz, et. al., "Wireless sensor networks: a survey," Computer Networks, Vol. 38, March 2002, pp.393-422.

[2] Stephen S.Yau, Fariaz Karim, Yu Wang, Bin Wang, and Sandeep K. S. Gupta, "Reconfigurable Context Sensitive Middleware for Pervasive Computing," IEEE Pervasive Computing, joint special issue with IEEE Personal Communications on Context-Aware Pervasive Computing IEEE Computer Society Press, Los Alamitos, USA, 1(3), July-September 2002, pp.33-40.

[3] J.Keeney, V.Cahill, "Chisel: A Policy-Driven, Context-Aware, Dynamic Adaptation Framework," Fourth IEEE International Workshop on Policies for Distributed Systems and Networks 'POLICY 2003', Italy, June 2003.

[4] Rodney Brooks. "The Intelligent Room Project," 2nd Int. Cognitive Technology Conference, Aizu, Japan, 1997.

[5] Steenkiste, P., "Aura: Invisible Ubiquitous Computing," Computer Systems Seminar, ETH Zurich, October 2002.

[6] R. Koodli and C. E. Perkins, "Service Discovery in On-Demand Ad Hoc Networks," Internet draft, MANET Working Group.

[7] A. Rakotonirainy, J. Indulska, Seng Wai Like, Arkady B.Zaslavsky, "Middleware for Reactive Components: An Integrated Use of Context, Roles and Event Based Coordination," IFIP/ACM International Conference on Distributed Systems Platforms, Middleware 01. (LNCS Vol. 2218), 2001, pp.77-98.
draft-koodli-manet-servicediscovery-00.txt, Sep. 2002.

[8] HP cooltown project http://www.cooltown.com/cooltown/

[9] Stanford, Interactive Workspaces Project http://iwork.stanford.edu/

[10] Microsoft, Easy Living Project http://research.microsoft.com/easyliving/

[11] Java Technology & XML – Documentation http://java.sun.com/xml/reference/docs/index.html

[12] Bo-seong.K, Young.Ko, "Implementation of service discovery on routing layer in Ad-hoc network environment," The Korean Information Science Society ,Vol. 31, September 2004.

[13] Arsalan Minhas, "Adaptive Middleware for Ubiquitous Computing," Tech. Report, Ajou University, June 2004.

[14] L. Capra, W. Emmerich and C. Mascolo., "CARISMA: Context-Aware Reflective mIddleware System for Mobile Applications" IEEE Transactions on Soft Eng, Volume 29, Num. 10, 2003, pp.929- 945.

[15] F. Kon, J.R. Marques, T. Yamane, R.H. Campbell, and M.D. Mickunas, "Design, implementation, and performance of an automatic con_guration service for distributed component systems" Software: Practice and Experience, 35(7), May

Bo-Seong Kim
He works for Samsung Electronic Company. He received the B.S. and M.S. degree in Information and Communication Engineering, Ajou University, Suwon, South Korea, in 2004 and 2006, respectively. His main research interests include system software, distributed system and ad hoc network.

Byoung-Hoon Lee

He received the B.S. and M.S. degree in Computer Engineering, Chungbuk National University, Cheongju, South Korea, in 1998 and 2000, respectively.

He is a graduate student of Graduate School of Information and Communication at Ajou University, Korea. His main research interests include ubiquitous computing, distributed system and embedded programming

Jai-Hoon Kim

He received the B.S. degree in Control and Instrumentation Engineering, Seoul National University, Seoul, South Korea, in 1984, M.S. degree in Computer Science, Indiana University, Bloomington, IN, U.S.A., in 1993, and his Ph.D. degree in Computer Science, Texas A&M University, College Station, TX., U.S.A., in 1997. He is currently an associate professor of the Information and Communication department at Ajou University, South Korea. His research interests include Distributed Systems, Real-Time Systems, Mobile Computing and Ubiquitous Computing.